Allow Me to Introduce
Spectral and Isoperimetric Graph Partitioning

DRAFT: Please add your name to the Acknowledgments by sending jrs comments and corrections at cs.berkeley.edu

Jonathan Richard Shewchuk

April 16, 2016

Department of Electrical Engineering and Computer Sciences
University of California at Berkeley
Berkeley, California 94720

Abstract

Graph partitioning is the problem of cutting a graph into two or more subgraphs of specified sizes while minimizing the total weight of the edges cut or a similar objective function. *Spectral partitioning* algorithms find approximate solutions to graph partitioning problems by computing one or more eigenvectors of a symmetric matrix that represents the connectivity of the graph, then inferring a cut from the eigenvectors. Spectral partitioning has a rich mathematical theory, both for deriving guaranteed bounds on the quality of spectral partitions and for creating *expander* graphs that are useful because they do not have good partitions. *Isoperimetric partitioning* is an alternative algorithm that has similar foundations in linear algebra, but it is faster and simpler because it replaces the eigenvalue computation with the solution of a symmetric linear system. However, it lacks the theoretical guarantees of spectral partitioning and it is too new to have been tested in a wide variety of applications.

The first half of this report describes the spectral and isoperimetric graph partitioning algorithms and the virtues and limitations of spectral partitioning as an approximation algorithm. The second half is a miscellany of variations and implications of these algorithms. Topics include Cheeger's inequality, the normalized cut criterion, vertex separators, negative edge weights, eigenvector computations, singular value decompositions for bipartite graphs, the use of multiple eigenvectors, and partitioning a graph into more than two subgraphs.

Supported in part by the National Science Foundation under Awards CCF-0430065, CCF-0635381, IIS-0915462, and CCF-1423560, in part by the University of California Lab Fees Research Program, and in part by an Alfred P. Sloan Research Fellowship. The claims in this document are those of the author. They are not endorsed by the sponsors or the U.S. Government.

Keywords: spectral graph partitioning, isoperimetric graph partitioning, Laplacian matrix, Fiedler vector, algebraic connectivity, edge separator, vertex separator, Cheeger's inequality, uniform sparsest cut, normalized cut, cut ratio, isoperimetric ratio, isoperimetric number, edge expansion, conductance, spectral clustering, vector partitioning, Lanczos algorithm

Contents

About this Report

These lecture notes are a sequel to my technical report *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain* (available from my web page), in the sense that I do not reprise material here that I cover there. There are a few sections of that report that might make this one easier to read. If you, like me, got through your undergraduate linear algebra class with no intuitive or visual understanding of eigenvectors, I suggest Section 5 of that report. Seeing eigenvectors as actors with a distinctive body language will open up spectral graph partitioning in a way not otherwise possible. Section 3 of that report offers a visual introduction to the quadratic form. Section 7.1 visually illustrates the meaning of M-orthogonality. Section 7.2 explains Gram–Schmidt conjugation. Appendix C2 has a proof that a symmetric matrix has n orthogonal eigenvectors. Finally, the conjugate gradient method itself is useful for isoperimetric graph partitioning.

I thank Gilbert Bernstein, Marc Khoury, and Darsh Ranjan for proofreading a draft. I thank Inderjit Dhillon, Bruce Hendrickson, Ravi Kolluri, James Lee, Gary Miller, James O'Brien, Horst Simon, Shang-Hua Teng, Luca Trevisan, and Stella Yu for discussions that clarified my understanding of spectral graph partitioners. I thank Dan Spielman, whose lecture notes filled many gaps, and (again) Luca Trevisan, whose lecture notes provided the proof of Cheeger's inequality. And I thank Dylan Scott and Maurya Talisetti, whose Mathematica code rendered some of the figures.

Part I

Spectral and Isoperimetric Graph Partitioning

1 Graph Partitioning, Linear Algebra, and Constrained Optimization

1.1 Graph Partitioning

The goal of *graph partitioning* is to cut a weighted, undirected graph into two or more subgraphs that are roughly equal in size, so that the total weight of the cut edges is as small as possible.

There are many variants of the graph partitioning problem. The simplest variant requests a bisection of a graph G into two subgraphs G_1 and G_2 that have an equal number of vertices, as illustrated at upper left in Figure 1, or differ in size by at most one vertex. A more general variant assigns a mass to each vertex, and asks that the sum of the vertex masses in G_1 be approximately equal to the sum of the vertex masses in G_2. Most of the uses of graph partitioning are not so demanding that they require the subgraphs to be perfectly balanced; for instance, it might suffice that the larger piece be no more than twice as big as the smaller. In some graphs, this looser restriction makes it possible to find a much smaller cut. However, the constraints should not be too liberal: if you simply ask for the smallest cut possible, a partitioner might slice just a single vertex from the rest of the graph, as illustrated at upper right in Figure 1. Thus, some versions of the graph partitioning problem trade off the weight of the cut against the imbalance of the subgraphs, as illustrated at lower left.

Graph partitioning can be formalized as a constrained optimization problem. You may choose among several objective functions, such as the total cut weight or the ratio of cut weight to lightest subgraph. Most of them are NP-hard to optimize.

This report summarizes two ways of reducing graph partitioning to problems in linear algebra that can be solved by effective, well-known algorithms. *Spectral graph partitioning* reduces graph partitioning to finding an eigenvector of a symmetric matrix. *Isoperimetric graph partitioning* reduces graph partitioning to solving a symmetric linear system.

Neither of these methods is a true reduction, though, as both reductions use the techique of *relaxing* a binary-valued unknown to a real-valued unknown. Ordinarily, a partition assigns each vertex of G to one subgraph, either G_1 or G_2. By contrast, the relaxed graph partitioning problem makes vertices divisible—it might assign two thirds of one vertex to G_1, and the other third to G_2. It might even assign three halves of one vertex to G_1, and minus one half to G_2. Relaxation turns a combinatorial optimization problem into a numerical optimization problem. The relaxed problems can be solved by polynomial-time algorithms.

Having solved a relaxed problem, you convert the real-valued solution to an approximately optimal solution of the original graph partitioning problem by *rounding* the vertex values—assigning each vertex to G_1 or G_2 by comparing it with some threshold value. The obvious threshold is to assign a vertex to G_1 if more than half of it is in G_1, or to G_2 otherwise, but that might give you a very unbalanced cut. If you prefer a partition that is as balanced as possible, place the threshold at the median vertex.

By rounding an optimal solution of a continuous problem, you produce a discrete partition, which you hope retains some semblance of near-optimality. An attractive feature of spectral partitioning is that a bound called *Cheeger's inequality* guarantees that you can always find a decently good cut. But "good" may fall well short of optimal, and well short of perfectly balanced. In practical applications, we are usually pleased, but there are graphs for which the spectral bisector is almost as bad as possible.

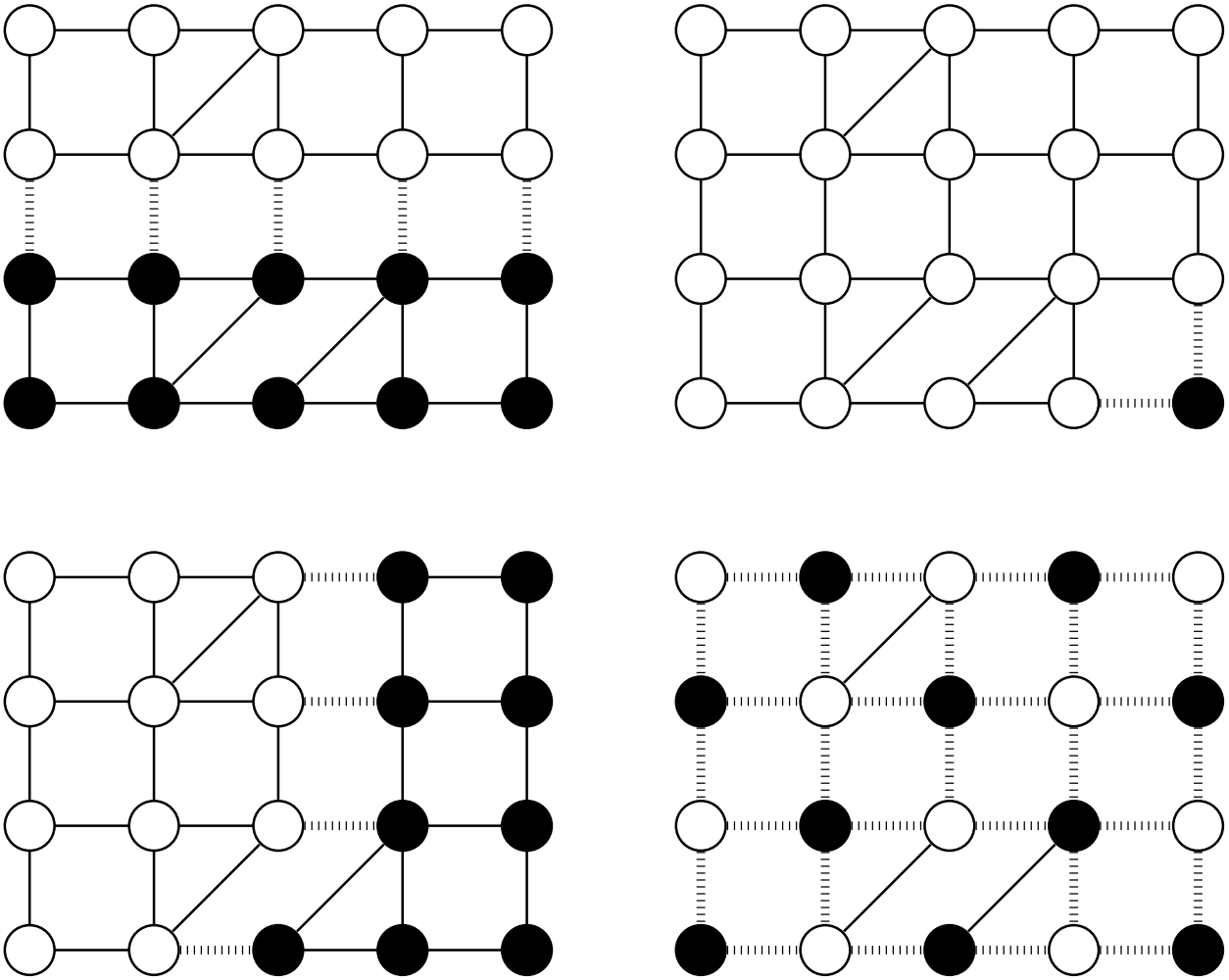

Figure 1: Several ways to cut a graph whose edge weights and vertex masses are all 1. The minimum bisection, at upper left, comprises five edges. The minimum cut, at upper right, comprises two edges but detaches just one vertex. The maximum cut, at lower right, comprises all but three edges, and for this graph also happens to be the maximum bisection. The sparsest cut, at lower left, has a sparsity of $4/(11 \cdot 9)$, and for this graph also has the minimum isoperimetric ratio, $4/9$.

Spectral partitioning was introduced by Kenneth Hall [31] in 1970 (in the setting of clustering; he didn't use the word "partition"), rediscovered by Wilm Donath and Alan Hoffman [17] in 1972 and by Miroslav Fiedler [23] in 1975, and popularized by Pothen, Simon, and Liou [58] in 1990.[1] It has become a mainstay of scientific computing and image processing. Isoperimetric partitioning was introduced by Leo Grady and Eric Schwartz [29] in 2006, and it appears to be a faster and easier alternative. Time will tell whether it will prove to be as effective or as popular as spectral partitioning.

Graph partitioners have many applications. They assign computations to processors in parallel computers [33, 58, 62, 68]: vertex masses represent the running times of computations, and cut weights represent communication times between processors. They *segment* images—that is, they find boundaries separating objects in photographs or medical images [61, 69, 70]. They are used in divide-and-conquer graph algo-

[1] Many publications credit the 1973 articles by Donath and Hoffman [18] and Fiedler [22], but a careful reading of those papers reveals no hint of a spectral partitioning algorithm. Sections 1.5, 2.7, and E.3 mention a few of those two papers' accomplishments.

rithms such as *nested dissection* [47], which orders the rows and columns of a sparse matrix to reduce the fill (nonzero components) in its Cholesky or LU decomposition. They cluster words that are frequently found together in documents, and cluster documents that have similar words [12]. Many of the earliest graph partitioning algorithms were motivated by circuit layout (on circuit boards and in integrated circuits), for which designers seek partitions of circuit components that minimize the number of long interconnections between physically distant components [2, 17, 31, 38].

In addition to graph partitioning algorithms, this report surveys a touch of spectral graph theory, notably Cheeger's inequality, which bounds the quality of the optimal cut (as measured by an objective function called the isoperimetric ratio) in terms of a graph eigenvalue. This theory's influence extends well beyond cutting graphs. The mixing times of random walks on graphs and the running times of approximate counting algorithms are governed by the optimal cut [63]. Graphs with no good partition, called *expanders*, are used in robust network design and error-correcting codes. Expanders are central in several results in complexity theory, including the complexity class equivalence SL = L (symmetric log-space = log-space, meaning that undirected graph reachability is solvable in logarithmic space) [59] and a proof of the famous PCP Theorem [14], which characterizes the complexity class NP in terms of probabilistically checkable proofs.

1.2 Definitions

Let $G = (\mathcal{V}, \mathcal{E})$ be an undirected graph with vertex set $\mathcal{V}$ and edge set $\mathcal{E}$. A *subgraph* of G is a graph obtained from G by removing a subset of its vertices and all the edges adjoining them (but leaving intact every edge whose endpoints both remain in the subgraph). A *partition* of G is a set of subgraphs G_1, G_2, ..., G_j such that every vertex in $\mathcal{V}$ appears in exactly one of the subgraphs.

Given a partition of G, the *cut* or *edge separator* is the set of edges in $\mathcal{E}$ whose endpoints lie in different subgraphs. We say that the partition *cuts* those edges. (Another type of separator, the *vertex separator*, is discussed in Section C.) Each edge has a specified numerical *weight*, which indicates how bad it is to cut that edge. The *weight* of a cut is the sum of the weights of all the edges in the cut. We wish to minimize this weight.

Most of this report addresses a *bipartition* of G into two subgraphs. *Multipartitions* into three or more subgraphs earn a lot of ink in Section E. A *bisection* is a perfectly balanced bipartition, and a *multisection* is a perfectly balanced multipartition. Most applications can benefit by sacrificing perfect balance to obtain a smaller cut; weakly balanced bipartitions are discussed starting in Section 2.2.

To influence how a partition should be balanced, we can assign each vertex a *mass*. The mass of a subgraph is the sum of the masses of its vertices. We might request that the subgraphs have equal mass (a *multisection*, which isn't always possible), or that their masses differ by at most a factor of two (which also isn't always possible). As linear algebraic partitioners operate by rounding a fractional solution, we can choose a rounding method that minimizes the difference between the subgraphs' masses. Alternatively, we can look for a partition that optimizes some objective function that trades cut weight against mass imbalance.

Two objective functions for judging unbalanced partitions are popular, in part because they emerge naturally from the mathematics of spectral graph theory. The *isoperimetric ratio* of a bipartition is

$$\frac{\text{Cut}(G_1, G_2)}{\min\{\text{Mass}(G_1), \text{Mass}(G_2)\}},$$

where $\text{Cut}(G_1, G_2)$ is the weight of the cut severing G_1 from G_2 and $\text{Mass}(G_i)$ is the total mass of the vertices in G_i. The minimum isoperimetric ratio among all cuts of G is called the *isoperimetric number* of G.

If every vertex has mass 1, thus $\text{Mass}(G_i) = |\mathcal{V}_i|$, the isoperimetric number is also called the *edge expansion*. If each vertex has mass equal to the sum of the adjoining edge weights (as described in Section A), the isoperimetric number is also called the *conductance* or the *Cheeger constant* of G. The graph in Figure 1 has an isoperimetric number of $4/9$, obtained by the cut at lower left.

The *sparsity* of a bipartition, also known as its *uniform sparsity* or *cut ratio*,[2] is

$$\frac{\text{Cut}(G_1, G_2)}{\text{Mass}(G_1)\,\text{Mass}(G_2)}.$$

The cut with minimum sparsity is called the *(uniform) sparsest cut*. If each vertex mass is the sum of the adjoining edge weights, the sparsest cut is also called the *normalized cut*, and the sparsity is also called the *normalized cut criterion* (see Section A). Figure 1 illustrates a cut of sparsity $1/20$ at upper left, and the sparsest cut, having sparsity $4/99$, at lower left.

The *average cut* criterion

$$\text{Cut}(G_1, G_2)\left(\frac{1}{\text{Mass}(G_1)} + \frac{1}{\text{Mass}(G_2)}\right)$$

is equal to the sparsity times $\text{Mass}(G)$, so optimizing one is equivalent to optimizing the other. Note that the average cut criterion is also equal to the sum of the isoperimetric ratios of the subgraphs G_1 and G_2.

This report focuses on NP-hard graph partitioning problems that are particularly well treated by spectral algorithms: finding the minimum bisection, the cut with minimum isoperimetric ratio, and the cut with minimum uniform sparsity. It neglects several graph partitioning problems that can be solved in polynomial time. For example, the *minimum cut problem*, illustrated at upper right in Figure 1, asks for the cut of minimum weight that disconnects the graph, though it might isolate just a single vertex. This problem can be solved in $O(|\mathcal{V}|\,|\mathcal{E}| + |\mathcal{V}|^2 \log |\mathcal{V}|)$ time [66]. In the classic *source-target minimum cut problem*, there are one or more *source vertices* that must be in G_1, and one or more *target vertices* that must be in G_2. The goal is to find the minimum cut that separates the source vertices from the target vertices. By the *max-flow min-cut theorem* [19, 24], this problem is equivalent to the *maximum network flow problem*, for which there is a rich literature of polynomial-time algorithms.

Most graph partitioning algorithms are formulated for graphs with positive edge weights; negative edge weights make many algorithms fail. For graphs with negative edge weights, the max-flow min-cut theorem does not hold, and both the minimum cut problem and the source-target minimum cut problem become NP-hard. Observe that finding the minimum cut of a graph with all edge weights negative is equivalent to finding the maximum cut of a graph with positive edge weights. Finding the *maximum cut*, illustrated at lower right in Figure 1, is one of the twenty-one problems that Karp [37] proved NP-complete in his famous paper that established the importance of the theory of NP-completeness.

An advantage of the spectral and isoperimetric partitioning algorithms is that they can accommodate negative edge weights; see Section G. Even though negative edge weights make some graph partitioning problems harder in principle, their careful use makes some applications faster and better.

1.3 The Laplacian Matrix and the Weight of a Cut

Linear algebraic methods for graph partitioning begin with the idea that we can express the weight of a cut in matrix notation as a quadratic function of binary variables.

[2] The uniform sparsest cut is a special case of the notion of a nonuniform sparsest cut in multicommodity flows. Uniformity arises when every vertex i wishes to route to every vertex j a flow proportional to $m_i m_j$, where m_i is the mass of vertex i. The term "cut ratio" refers to the amount of flow that can cross the cut divided by the amount of demand across the cut.

Number the vertices of $G = (\mathcal{V}, \mathcal{E})$ from 1 to $n = |\mathcal{V}|$. Let (i, j) denote an edge connecting vertex i to vertex j, and let w_{ij} be its weight, with $w_{ij} = 0$ if $(i, j) \notin \mathcal{E}$. As G is undirected, $(i, j) = (j, i)$ and $w_{ij} = w_{ji}$. Assume that there are no self-edges of the form (i, i).

Suppose we are given a partition of G into subgraphs G_1 and G_2. Let x be an n-component *indicator vector* that represents the partition as follows. Let c_1 and c_2 be two distinct, arbitrary constants. We set $x_i = c_1$ if vertex i is in subgraph G_1, and $x_i = c_2$ if vertex i is in subgraph G_2. The contribution of an edge (i, j) to the cut weight is

$$w_{ij}\frac{(x_i - x_j)^2}{(c_1 - c_2)^2} = \begin{cases} w_{ij}, & (i, j) \text{ is cut,} \\ 0, & (i, j) \text{ is not cut.} \end{cases} \tag{1}$$

Hence, the weight of the cut is

$$\begin{aligned} \text{Cut}(G_1, G_2) &= \sum_{(i,j)\in\mathcal{E}} w_{ij}\frac{(x_i - x_j)^2}{(c_1 - c_2)^2} && (2) \\ &= \frac{1}{(c_1 - c_2)^2}\sum_{(i,j)\in\mathcal{E}} \left(w_{ij}x_i^2 - 2w_{ij}x_ix_j + w_{ij}x_j^2\right) \\ &= \frac{1}{(c_1 - c_2)^2}\Bigg(\underbrace{\sum_{(i,j)\in\mathcal{E}} -2w_{ij}x_ix_j}_{\text{off-diagonal terms}} + \underbrace{\sum_{i=1}^{n} x_i^2 \sum_{k\neq i} w_{ik}}_{\text{diagonal terms}} \Bigg) \\ &= \frac{x^{\mathsf{T}}Lx}{(c_1 - c_2)^2}, && (3) \end{aligned}$$

where L is a symmetric $n \times n$ matrix called the *Laplacian matrix* for G, whose components are

$$L_{ij} = \begin{cases} -w_{ij}, & i \neq j, \\ \sum_{k\neq i} w_{ik}, & i = j. \end{cases} \tag{4}$$

L is effectively a matrix representation of G. (For the purpose of partitioning a graph, there is no need to distinguish edges of weight zero from edges that are not in the graph.)

Equation (3) shows that minimizing the weight of the cut is equivalent to minimizing $x^{\mathsf{T}}Lx$, the *Laplacian quadratic form*. This is a core observation underlying spectral and isoperimetric graph partitioners.

If every weight w_{ij} is nonnegative, Expression (2) implies that $x^{\mathsf{T}}Lx \geq 0$ for all x—said differently, L is positive semidefinite and thus L has no negative eigenvalues. Moreover, if G is a connected graph and all its edges have positive weight, then Expression (2) is zero if and only if all the components of x are equal to each other. Therefore, L has exactly one eigenvector with eigenvalue zero,[3] namely, $\mathbf{1} = [1, 1, \ldots, 1]^{\mathsf{T}}$. This eigenvector reflects the fact that the cut has weight zero if and only if every vertex is in one subgraph and the other subgraph is empty.

Figure 2 depicts isosurfaces of the form $x^{\mathsf{T}}Lx = c$ for a three-vertex graph. The isosurfaces are concentric elliptical cylinders whose central axis is parallel to $\mathbf{1}$.

If G is not connected, but all its edges have positive weight, then L has one zero eigenvalue for each connected component of G: setting $x_i = 1$ for each vertex i in a component and $x_i = 0$ for each vertex in the other components yields an eigenvector with eigenvalue zero. The central axis in Figure 2 is no longer a line, but rather a plane or higher-dimensional subspace spanned by these eigenvectors. There are other

[3] Recall that v is an eigenvector of L with eigenvalue λ if $Lv = \lambda v$.

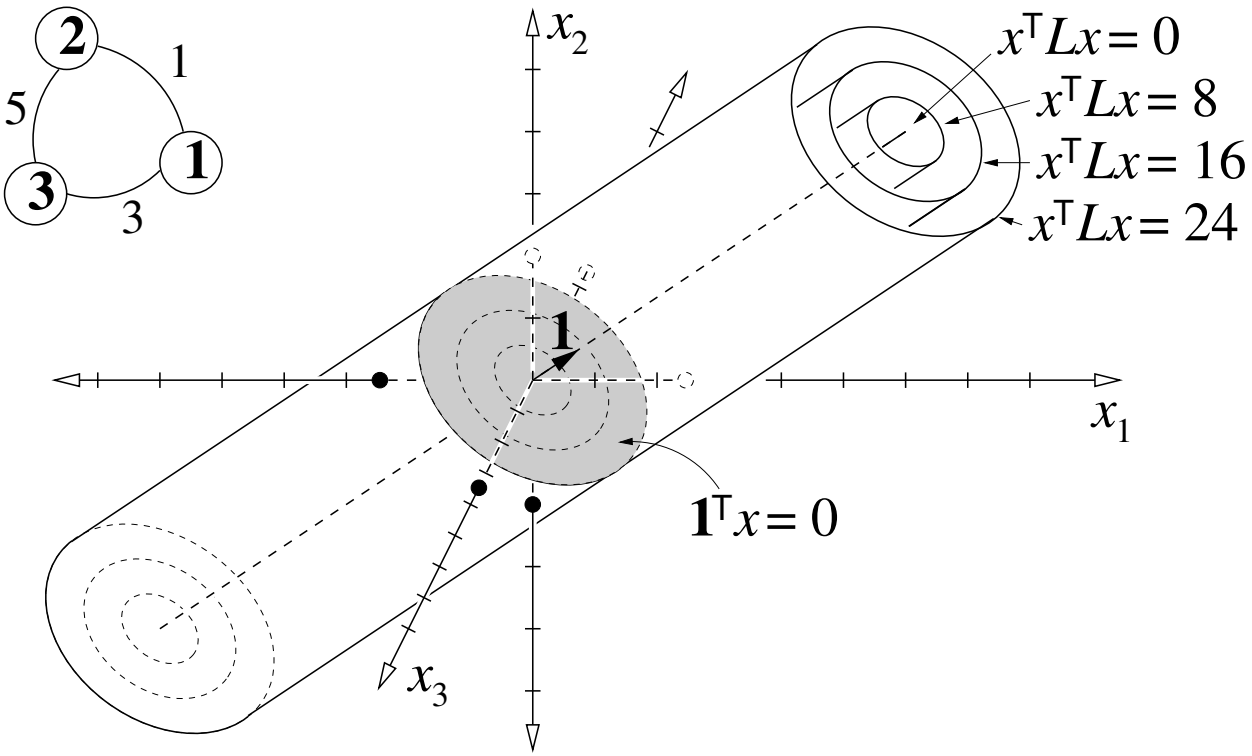

Figure 2: Isosurfaces of the function $x^T Lx$ for the graph at upper left. The shaded region represents the linear balance constraint $\mathbf{1}^T x = 0$.

eigenvectors with eigenvalue zero (including $\mathbf{1}$), but they are all linear combinations of these ones, because Expression (2) is zero if and only if x assigns all the vertices in a connected component the same value.

The foregoing discussion assumes that c_1 and c_2 are constant and distinct, but it holds regardless of their specific values. Their role will become nebulous in two ways. First, a partitioning problem is relaxed by permitting the components of x to take on any real value, not just c_1 or c_2. Second, c_1 and c_2 do not always need to be chosen in advance. In Section 2.6, we will choose them to be a function of the cut—specifically, a measure of the balance.

1.4 Graph Partitioning as Constrained Quadratic Optimization

The cut weight achieves its minimum value, $x^T Lx = 0$, when $G_1 = G$ and $G_2 = (\emptyset, \emptyset)$, but that's not useful. Hence, we impose a *balance constraint*.

Suppose we choose the constants $c_1 = 1$ and $c_2 = -1$ to signify which vertices are in G_1 and which are in G_2; thus we call x a *signed indicator vector*. (This name is in contrast to the *0-1 indicator vectors* used in isoperimetric partitioning and some spectral multipartitioning algorithms.) We can specify that exactly half the vertices are in G_1 and half are in G_2 by imposing the constraint

$$\mathbf{1}^T x = 0, \tag{5}$$

which dictates that the number of components of x that are 1 is equal to the number that are -1. Figure 2 illustrates this constraint as a shaded planar cross-section of the cylindrical isosurfaces. Of course, this

constraint can only be satisfied if the number n of vertices is even—but that will be no impediment once we relax the partitioning problem, so don't worry about it.

Thus, we reduce the graph bisection problem to the optimization problem of choosing x to

$$\begin{aligned} \text{minimize} \quad & x^\mathsf{T} L x \\ \text{subject to} \quad & \forall i, x_i = 1 \text{ or } x_i = -1 \\ \text{and} \quad & \mathbf{1}^\mathsf{T} x = 0. \end{aligned} \tag{6}$$

This problem is NP-complete [25]. Moreover, it is NP-hard to find a cut within a constant factor of the minimum, even if no vertex has degree greater than three, even if you relax (5) to permit a constant multiple of imbalance (e.g. if one subgraph is permitted to be at most 100 times larger than the other) [8].

What if you want a biased cut? For example, suppose you want G_1 to have exactly four times as many vertices as G_2. Although your first impulse might be to revise the balance constraint, spectral partitioning naturally enforces (5) exactly as written. A more compatible way to enforce the constraint is by choosing $c_1 = 1$ and $c_2 = -4$—that is, every component of x is constrained to be -4 or 1—while leaving the balance constraint (5) unchanged. This trick has surprisingly broad implications, which Sections 2.6, 2.7, and E.5 explore.

1.5 Bounds on the Weight of a Bisection

The balance constraint (5) specifies that x be orthogonal to $\mathbf{1}$, which happens to be an eigenvector of L with eigenvalue zero, which is the smallest eigenvalue of L if no edge has negative weight. This fact implies that there is a relationship between the smallest balanced cut and the *second*-smallest eigenvalue of L. This relationship points directly to a spectral partitioning algorithm. Let's take a look.

Let $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$ be the eigenvalues of L (including multiplicities) in order from least to greatest. Let v_1, v_2, …, v_n be the corresponding *unit* eigenvectors. Thus, $\lambda_1 = 0$ and $v_1 = (1/\sqrt{n})\mathbf{1}$. Let $V = [v_1, v_2, \ldots, v_n] \in \mathbb{R}^{n \times n}$, and let Λ be the diagonal $n \times n$ matrix whose diagonal entries are the eigenvalues. By the definition of eigenvector, $Lv_i = \lambda_i v_i$, so

$$LV = V\Lambda.$$

As L is symmetric, its eigenvectors are all orthogonal to each other (except perhaps eigenvectors that have the same eigenvalue, as any linear combination of these is also an eigenvector; but assume we choose the v_i's to be mutually orthogonal). The columns of V form an orthonormal basis for $\mathbb{R}^n$. V is said to be an *orthonormal matrix*, meaning that $V^\mathsf{T} V = I$.

We seek an x that minimizes $x^\mathsf{T} L x$. If you are familiar with the *Rayleigh quotient* $x^\mathsf{T} L x \,/\, x^\mathsf{T} x$, you might suspect a connection. As each component of x is constrained to be 1 or -1, the denominator $x^\mathsf{T} x$ is always n, so minimizing the cut weight is equivalent to minimizing the Rayleigh quotient. By the well known *Rayleigh–Ritz theorem*, for every nonzero vector x, its Rayleigh quotient is in the range $[\lambda_1, \lambda_n]$; moreover, if x satisfies (5)—that is, if x is orthogonal to v_1—then the Rayleigh quotient is in the range $[\lambda_2, \lambda_n]$.

To prove that this is true, write x as a linear combination of the unit eigenvectors,

$$x = Va, \tag{7}$$

where $a \in \mathbb{R}^n$. As V is orthonormal, we can view V as a rotation (or reflection), and we can view a as the vector x rotated into the coordinate system of the eigenvectors. Rotation does not change the length of x; in other words,

$$n = x^\mathsf{T} x = a^\mathsf{T} V^\mathsf{T} V a = a^\mathsf{T} a.$$

The objective function is

$$\begin{aligned} x^\mathsf{T} L x &= a^\mathsf{T} V^\mathsf{T} L V a \\ &= a^\mathsf{T} V^\mathsf{T} V \Lambda a \\ &= a^\mathsf{T} \Lambda a \\ &= \sum_{i=1}^{n} \lambda_i a_i^2. \end{aligned} \tag{8}$$

Because x is orthogonal to $\mathbf{1}$, its expression as a linear combination of orthogonal eigenvectors does not include a contribution from $v_1 = \mathbf{1}$, so the coefficient a_1 is zero. To make this rigorous, observe that $a = V^\mathsf{T} V a = V^\mathsf{T} x$, so $a_1 = v_1^\mathsf{T} x = \mathbf{1}^\mathsf{T} x = 0$. Thus we drop the first term of the summation (8), which is therefore bounded by the inequalities

$$\lambda_2 \sum_{i=2}^{n} a_i^2 \le x^\mathsf{T} L x \le \lambda_n \sum_{i=2}^{n} a_i^2. \tag{9}$$

But $\sum_{i=2}^{n} a_i^2 = a^\mathsf{T} a = x^\mathsf{T} x$, verifying that the Rayleigh quotient is in $[\lambda_2, \lambda_n]$.

Substitution of $\sum_{i=2}^{n} a_i^2 = x^\mathsf{T} x = n$ into (9) and (9) into Equation (3), with $c_1 = 1$ and $c_2 = -1$, yields bounds on the weight of every bisection:

$$\frac{\lambda_2 n}{4} \le \mathrm{Cut}(G_1, G_2) \le \frac{\lambda_n n}{4}. \tag{10}$$

It is a bit magical that linear algebra can place bounds on a combinatorial optimization problem. In particular, the lower bound of $\lambda_2 n/4$ sometimes informs us that there is no small bisection, without our needing to solve the NP-hard problem of trying to find one. Donath and Hoffman [18] derived the lower bound (10) in 1973 (albeit for the special case where every edge has weight 1).

Suppose that the components of the vector $\sqrt{n} v_2$ all just happen to be ± 1, so the second eigenvector represents a bisection. (The coefficient $\sqrt{n}$ appears because v_2 is a unit vector but every indicator vector has length $\sqrt{n}$.) Substituting $x = \sqrt{n} v_2$ into the objective function $\mathrm{Cut}(G_1, G_2) = x^\mathsf{T} L x/4$ gives $\lambda_2 n/4$, which matches the lower bound. Therefore, in this lucky circumstance $\sqrt{n} v_2$ describes an optimal bisection!

The vector v_2 is often called the *Fiedler vector*[4], after Miroslav Fiedler, who suggested using it to partition graphs [23]. Sadly, to hope that every component of $\sqrt{n} v_2$ will be 1 or -1 is usually too much magic to ask for. But we will see that we can often find a good cut by rounding the Fiedler vector.

Likewise, λ_2 is occasionally called the *Fiedler value*, but Fiedler himself calls λ_2 the *algebraic connectivity* of G. In the special case where every edge has weight 1 and G is not a complete graph, Fiedler [22] showed in 1973 that the *vertex connectivity* of G—the minimum number of vertices we must remove to separate G into two or more connected components—is at least λ_2. The *edge connectivity* of G—the number of edges in the minimum cut (without any balance condition)—is never less than the vertex connectivity, and thus also is at least λ_2. (Compare with the bound of $\lambda_2 n/4$ for the minimum *balanced* cut.) Hence every vertex of G has degree at least λ_2. These relationships do not hold for weighted graphs, but we will see a slightly weaker inequality for the minimum cut weight of a weighted graph in Section 2.7. We will see in Section B that λ_2 also induces bounds—lower bounds and, more crucially, upper bounds—on G's isoperimetric number and sparsity.

[4]Fiedler calls v_2 the *characteristic valuation*, which was an excellent strategy for getting it renamed after him.

2 Spectral Graph Partitioning

2.1 The Relaxed Optimization Problem

Relax the optimization problem (6) by dropping the constraint that $x_i = \pm 1$, replacing it with the weaker constraint that x has length $\sqrt{n}$. The relaxed problem is

$$\begin{array}{rl} \text{minimize} & x^{\mathsf{T}} L x \\ \text{subject to} & x^{\mathsf{T}} x = n \\ \text{and} & \mathbf{1}^{\mathsf{T}} x = 0. \end{array} \tag{11}$$

The relaxation changes the domain of x from the corners of a hypercube to a hypersphere passing through those corners. The eigenvector $x = \sqrt{n} v_2$ is always a solution, because it achieves the lower bound (10).

Figure 3 illustrates the relaxed optimization problem for the three-vertex graph example. The diagram has you peering down the central axis of the cylindrical isosurfaces from Figure 2. The eigenvectors of L (except v_1) are the axes of the ellipses, and the eigenvalues determine their ellipticity. The quadratic constraint forces the solution to lie on a sphere, and the balance constraint forces the solution to lie on the page (representing the planar cross-section from Figure 2). The intersection of the two constraints is a circle. (If there were more than three vertices, it would be a hypersphere.) We seek the point on this circle that minimizes $x^{\mathsf{T}} L x$. A solution, $\sqrt{3} v_2 = (-1.366, 1, 0.366)$, appears as a black point. (Its negation, $-\sqrt{3} v_2$, is the only other solution.) The Fiedler vector cleanly separates vertex 1 from the other two vertices.

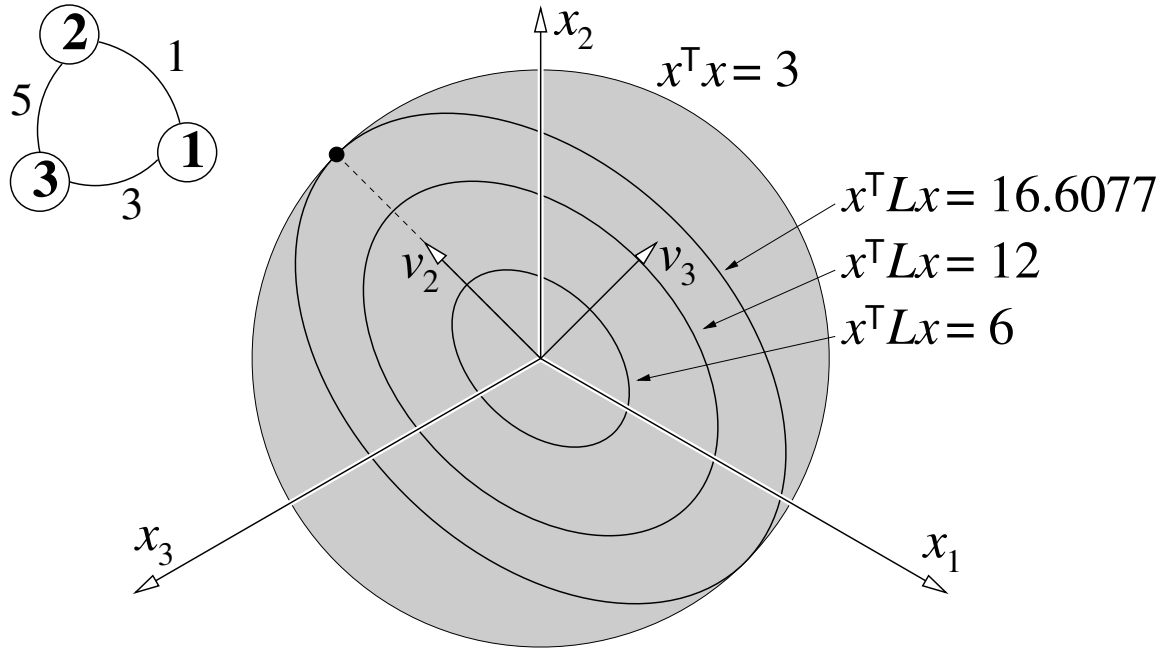

Figure 3: The view down the central axis of the cylindrical isosurfaces of $x^{\mathsf{T}} L x$ (recall Figure 2) for the graph at upper left. The eigenvector $v_1 = (1/\sqrt{3})\mathbf{1}$ points directly at you. Only the plane induced by the balance constraint $\mathbf{1}^{\mathsf{T}} x = 0$ is drawn. The large circle is the intersection of the balance constraint (a plane) with the quadratic constraint $x^{\mathsf{T}} x = 3$ (a sphere). The black dot is one of the two constrained minima.

Figure 4: The search spaces for four optimization problems (ignoring the balance constraint). From left to right: The domain of the discrete optimization problem is the vertices of a hypercube. The domain of the relaxed optimization problem solved by the simplest spectral partitioner is a hypersphere passing through those vertices. The domain of the relaxed problem solved by a spectral partitioner with vertex masses is an ellipsoid. The domain of the relaxed problem solved by an isoperimetric partitioner is a hyperplane.

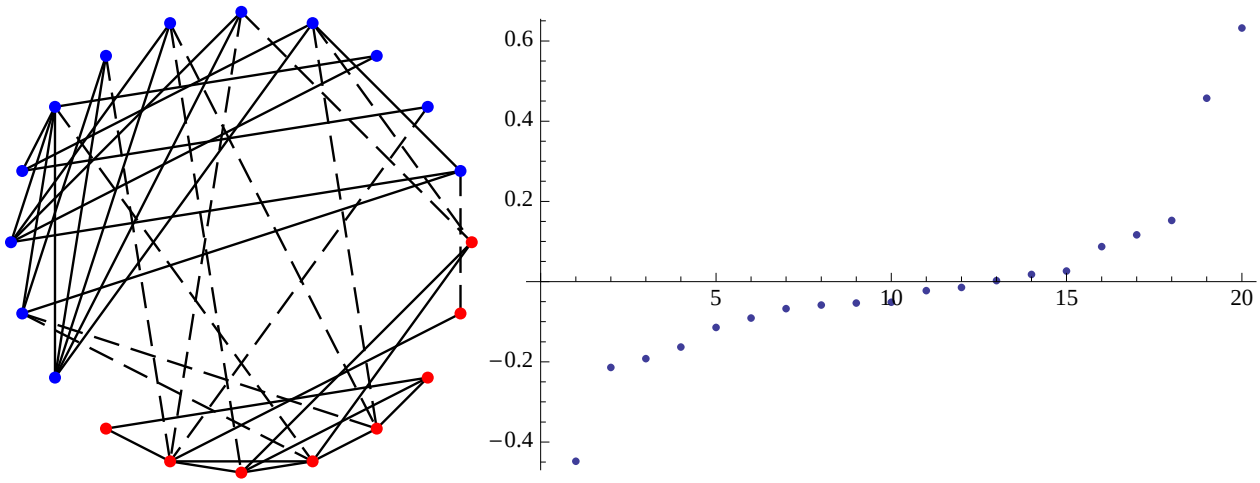

Figure 5: A graph and its Fiedler vector. All the edge weights and vertex masses are 1. The vertices have been sorted in clockwise order by increasing component in the Fiedler vector. The graph is partitioned by a sweep cut that finds the threshold offering the minimum sparsity.

There are two reasons to constrain the length of x in the relaxed problem (11): first, to prohibit the useless minimizer $x = \mathbf{0}$; second, as we saw in Section 1.5, the Fiedler vector naturally minimizes $x^T Lx$ subject to the constraint that x has fixed length.

This constraint is arbitrary; we choose it because it yields a problem that somebody knows how to solve. We treat vertex masses by replacing the spherical constraint with an ellipsoidal constraint (see Section 2.3). Isoperimetric partitioning is faster and simpler simply because it replaces the quadratic constraint with a linear constraint that is easier to satisfy (see Section 3.2). Figure 4 illustrates the different relaxations.

Curiously, the relaxed problem retains no memory of whether we want a balanced cut or a biased cut, because c_1 and c_2 do not appear in the problem (11). This might seem like a shortcoming, but it's really a virtue. Only when we round the solution vector do we need to decide how we want to balance the partition.

2.2 The Spectral Partitioning Algorithm (without Vertex Masses)

Given a connected, weighted graph G, construct its Laplacian matrix L and compute an eigenvector v_2 of L associated with L's second-smallest eigenvalue (by methods described in Section H). Typically, the components of v_2 are not two-valued, but are distributed somewhat smoothly over a range of real values, as illustrated in Figure 5. Thus, we *round* the components of the eigenvector. There are several rounding heuristics to choose from, but nearly all the common heuristics are *threshold cuts*: they choose a threshold value that separates the vertices assigned to G_1 from the vertices assigned to G_2.

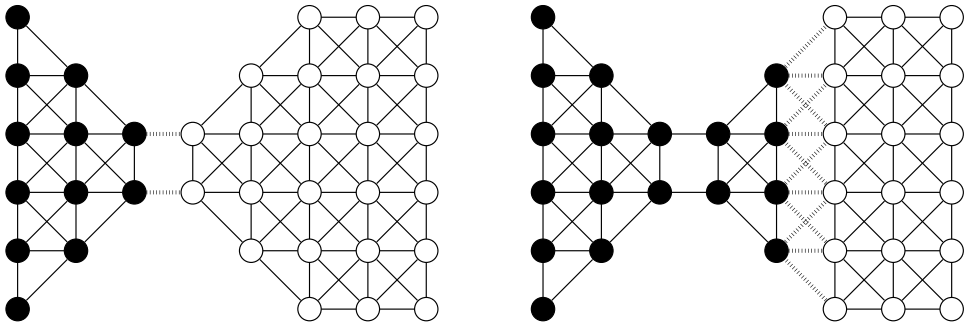

Figure 6: A graph whose sparsest cut (left) is dramatically better than its minimum bisection (right).

The simplest rounding method is the *sign cut* or *zero threshold cut*: assign vertex i to G_1 if component i of v_2 is positive, and to G_2 otherwise. Fiedler [23] proves that if no component of v_2 is exactly zero, then each of G_1 and G_2 is connected. (If some component is exactly zero, G_2 is connected—presuming it contains all the vertices with value zero—but G_1 might not be.)

Rounding by sign does not usually produce a balanced cut. The balance constraint (5) ensures that the *mean* component of x is zero, but as Figure 5 illustrates, it does not ensure that there are equal numbers of positive and negative components.

We can guarantee perfect balance by finding the median component, then using it as the threshold for bisecting the graph. (If several vertices are assigned a value equal to the median component, divide them between the subgraphs so that the cut is balanced.) This partition is called the *median cut*. See Sections B.2, D, and F for alternative bisection methods that are not threshold cuts.

Near-perfect balance is essential for some applications, such as the assignment of parallel computations to processors, but applications that tolerate imperfect balance have an advantage. Some graphs admit a much smaller cut if a little imbalance is permitted, like the graph in Figure 6, and many divide-and-conquer algorithms such as nested dissection [47] can benefit from that leeway.

The most popular way to choose a threshold is called the *sweep cut* or *criterion cut*. Begin by sorting the components of v_2. Then try out every feasible threshold (i.e. cutting between each pair of successive components) by explicitly computing the cut weight if that threshold is chosen. Choose the threshold that delivers the smallest cut. Because the cut changes incrementally as you sweep through the list of sorted components, this search can be done in $O(|\mathcal{V}| + |\mathcal{E}|)$ time (with standard sparse matrix representations).

The sweep cut can employ criteria other than the cut weight. For example, you can choose the isoperimetric ratio, the sparsity, or the normalized cut criterion, all defined in Section 1.2. (Figure 5 illustrates a sweep cut with the unit-mass sparsity as its criterion.) These choices trade cut weight against subgraph imbalance in different ways.

A related rounding method is to sort the components and look for the largest gap between two successive values (perhaps restricted to the middle third of the sorted list for balance). This partition is called the *jump cut* or *gap cut*. It is far less reliable than the sweep cut, but it is easier to program.

A graph that is not connected deserves special treatment. If G has k connected components, the k smallest eigenvalues of L are zero and the corresponding eigenvectors distinguish among the connected components, but they do not differentiate vertices within each component. Subsequent eigenvectors could be used to cut individual components (see Section D), but it is wiser to separate G into its connected components and partition just one of them.

Clustering is a form of graph partitioning in which the sizes of the subgraphs—and often the number of subgraphs—are not known in advance, but are induced by the data. It works best when a graph can be partitioned into subgraphs whose internal connectivity is much stronger than the connectivity between subgraphs. Clustering is appropriate for image segmentation, document analysis, and many tasks in artificial intelligence and statistical analysis.

Clustering typically begins with a weighted graph that represents the strengths of relationships, often called *affinities* or *similarities*, within a set of objects. One way to cluster the objects is to compute the Fiedler vector and partition its components with a one-dimensional spatial clustering algorithm, such as Lloyd's algorithm for k-means clustering [49]. As Section E details, another way is to use multiple eigenvectors to assign each object a point in a low-dimensional space, then apply Lloyd's algorithm to the points in that space.

2.3 Vertex Masses

What if not all vertices are equal? Sometimes the notion of balance must accord more prominence to some vertices than others, just as some edges are more expensive to cut. For example, if our goal is to partition computations among parallel processors, we wish to assign each processor an equal amount of computation time. Some vertices might represent more computation than others.

Suppose each vertex i has a specified positive *mass* m_i. The *mass* of a graph G, denoted $\text{Mass}(G)$, is the sum of its vertices' masses. The goal is to cut G into two subgraphs G_1 and G_2 of roughly equal mass.

Define a diagonal *mass matrix* M whose diagonal components are $M_{ii} = m_i$. As before, the signed indicator vector x represents a partition by having $x_i = 1$ if vertex i is in G_1, and $x_i = -1$ if vertex i is in G_2. The balance constraint is now

$$\mathbf{1}^\mathsf{T} M x = 0. \tag{12}$$

Recall that when we solve the relaxed optimization problem (11), we don't enforce the balance constraint (5) explicitly; rather, it is enforced implicitly when we bypass the smallest eigenvalue of L for the second-smallest. How do we replace the old balance constraint with the new one?

Van Driessche and Roose [68] observe that an elegant way to do so is to choose a different quadratic constraint for the relaxed problem, namely, $x^\mathsf{T} M x = \text{Mass}(G)$. The new relaxed problem is to

$$\begin{aligned} \text{minimize} \quad & x^\mathsf{T} L x \\ \text{subject to} \quad & x^\mathsf{T} M x = \text{Mass}(G) \\ \text{and} \quad & \mathbf{1}^\mathsf{T} M x = 0. \end{aligned} \tag{13}$$

The relaxation changes the domain of x from the corners of a hypercube to an ellipsoid (rather than a hypersphere) passing through those corners, as illustrated in Figure 4.

To solve the problem (13), compute the eigenvector v_2 associated with the second-smallest eigenvalue of the symmetric *generalized eigensystem* $Lv = \lambda Mv$, and scale the eigenvector to satisfy the ellipsoid constraint.[5] The idea to use this generalized eigensystem to account for vertex masses dates back to the 1972 paper of Donath and Hoffman [17] (though they do not present their method as a relaxation of a discrete optimization problem).

[5] A necessary condition for x to be a solution of the constrained optimization problem (13) is that $\nabla(x^\mathsf{T} L x) = \lambda \nabla(x^\mathsf{T} M x)$ for some real-valued Lagrange multiplier λ. Taking the gradients, we have $Lx = \lambda Mx$.

Observe that $\mathbf{1}$ is an eigenvector of the generalized eigensystem too, because $L\mathbf{1} = \mathbf{0}$. However, the eigenvectors of a generalized eigensystem are not (generally) orthogonal to each other; rather, they are *M-orthogonal* to each other, meaning that $v_i^\mathsf{T} M v_j = 0$ when $i \neq j$. Therefore, any linear combination of the eigenvectors excluding $\mathbf{1}$ satisfies the balance constraint (12).

Let $M^{1/2}$ be the diagonal matrix whose diagonal components are the positive square roots of the corresponding components of M; thus $(M^{1/2})^2 = M$. The eigenvalues of the generalized eigensystem $Lv = \lambda Mv$ are the same as the eigenvalues of the *generalized Laplacian matrix* $M^{-1/2}LM^{-1/2}$. Thus, the eigensystem is positive semidefinite and has no negative eigenvalues. However, the eigenvectors are not the same. If v is an eigenvector of $Lv = \lambda Mv$, then $M^{1/2}v$ is an eigenvector of $M^{-1/2}LM^{-1/2}$ associated with the same eigenvalue.

In some applications, such as image segmentation algorithms and the analysis of random walks on graphs, each vertex's mass is the sum of the adjoining edge weights—that is, $M_{ii} = L_{ii}$ (see Section A). In this case, the matrix $M^{-1/2}LM^{-1/2}$ is called the *normalized Laplacian*, because its diagonal entries are all 1.

Let us revisit the bounds from Section 1.5 in the light of vertex masses. The analysis is almost the same, but we must take care of the details to make sure that the masses appear in the right places. This time, λ_1, $\lambda_2, \ldots, \lambda_n$ and $v_1, v_2, \ldots, v_n$ are the eigenvalues and eigenvectors of the generalized eigensystem $Lv = \lambda Mv$. As before, let $V = [v_1, v_2, \ldots, v_n]$, and let Λ be the diagonal $n \times n$ matrix whose diagonal entries are the eigenvalues. Because $Lv_i = \lambda_i M v_i$, we have $LV = MV\Lambda$.

Instead of using unit eigenvectors, scale the eigenvectors so that $v_i^\mathsf{T} M v_i = 1$. Because the eigenvectors are mutually M-orthogonal, $V^\mathsf{T} MV = I$.

The *generalized Rayleigh quotient* is $x^\mathsf{T} Lx / x^\mathsf{T} Mx$. For every $x \neq \mathbf{0}$ that satisfies (12), this Rayleigh quotient is in the range $[\lambda_2, \lambda_n]$. To see this, write x again as a linear combination $x = Va$ of the eigenvectors. We are minimizing

$$\begin{aligned} x^\mathsf{T} Lx &= a^\mathsf{T} V^\mathsf{T} LVa \\ &= a^\mathsf{T} V^\mathsf{T} MV\Lambda a \\ &= a^\mathsf{T} \Lambda a \\ &= \sum_{i=1}^{n} \lambda_i a_i^2. \end{aligned}$$

Observe that $a = V^\mathsf{T} MVa = V^\mathsf{T} Mx$ and hence $a_1 = v_1^\mathsf{T} Mx = \mathbf{1}^\mathsf{T} Mx = 0$. We can drop the first term of the summation, which is then bounded by the inequalities

$$\lambda_2 \sum_{i=2}^{n} a_i^2 \leq x^\mathsf{T} Lx \leq \lambda_n \sum_{i=2}^{n} a_i^2. \tag{14}$$

But $x^\mathsf{T} Mx = a^\mathsf{T} V^\mathsf{T} MVa = a^\mathsf{T} a = \sum_{i=2}^{n} a_i^2$, verifying that the generalized Rayleigh quotient is in $[\lambda_2, \lambda_n]$.

Substituting $\sum_{i=2}^{n} a_i^2 = x^\mathsf{T} Mx = \text{Mass}(G)$ into (14) and (14) into (3), we obtain the bounds

$$\frac{\lambda_2 \,\text{Mass}(G)}{4} \leq \text{Cut}(G_1, G_2) \leq \frac{\lambda_n \,\text{Mass}(G)}{4}. \tag{15}$$

It follows that the eigenvector $x = \sqrt{\text{Mass}(G)}\, v_2$ is a solution of the optimization problem (13), because substitution gives $\text{Cut}(G_1, G_2) = x^\mathsf{T} Lx/4 = \lambda_2 \,\text{Mass}(G)/4$, which matches the lower bound. If all the components of $\sqrt{\text{Mass}(G)}\, v_2$ just happen to be ± 1, it describes a minimum bisection. Likewise, the upper bound implies that $\sqrt{\text{Mass}(G)}\, v_n$ is a maximum of the relaxed optimization problem, and if all its components are ± 1, it describes a maximum bisection (and as Section 2.7 will show, a maximum cut). These facts justify the use of v_2 (or v_n) for partitioning.

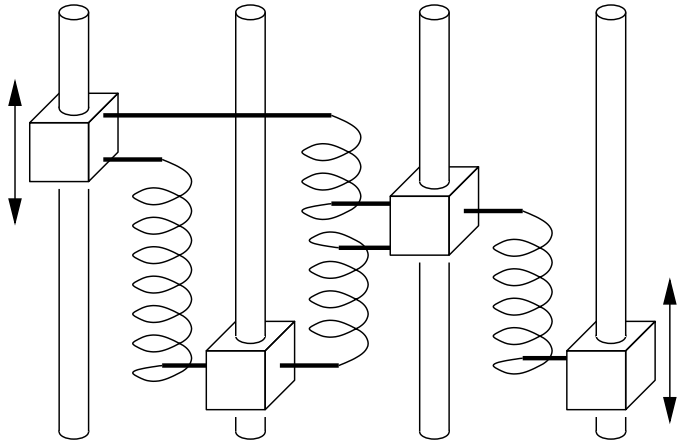

Figure 7: The eigensystem models vibrational modes in a system of springs and masses.

2.4 The Spectral Partitioning Algorithm with Vertex Masses

Given a connected, weighted graph G and the vertex masses, construct G's Laplacian matrix L and the diagonal mass matrix M. Compute an eigenvector v_2 associated with the second-smallest eigenvalue of the generalized eigensystem $Lv = \lambda Mv$. (One way to do that is to compute the corresponding eigenvector of the generalized Laplacian matrix $M^{-1/2}LM^{-1/2}$, then premultiply it by $M^{-1/2}$.) Round the eigenvector as described in Section 2.2.

2.5 The Vibration Analogy

To gain intuition about spectral partitioning, think of the eigenvectors as vibrational modes in a physical system of springs and masses, illustrated in Figure 7.

Each vertex models a point mass m_i that is constrained to move freely along a vertical rod. Each edge models a vertical spring with rest length zero and stiffness proportional to its weight, pulling two point masses together. The masses are free to oscillate sinusoidally on their rods. The eigenvectors of the generalized eigensystem $Lv = \lambda Mv$ are the vibrational modes of this physical system, and their eigenvalues are proportional to their frequencies.

Figure 8 illustrates the first four eigenvectors for two simple graphs. The first eigenvector, $v_1 = \mathbf{1}$, represents a vertical translation of all the masses in unison—which isn't really a vibration, as the eigenvalue of zero indicates. The second eigenvector v_2 represents the vibrational mode with the lowest frequency. Each component of v_2 indicates the amplitude with which the corresponding point mass oscillates. At any point in time as the masses vibrate, roughly half the mass is moving up while half is moving down. A sign cut cleaves the former mass from the latter. For the examples in Figure 8, the sign cut is an optimal bisection. The third eigenvector v_3 also induces a reasonable bisection of the grid graph, entirely different from the bisection induced by v_2.

To see why low-frequency modes of vibration induce good partitions, it is more intuitive to consider a continuous physical system, rather than a discrete one (especially when a spectral graph partitioner is used to parallelize the simulation of a physical system). Imagine you are an astronaut with a drumstick, tapping a cymbal floating weightlessly inside your spaceship. The taps excite vibrational modes of the cymbal, which you hear as harmonics of various pitches. As an approximation, think of the cymbal as a two-dimensional

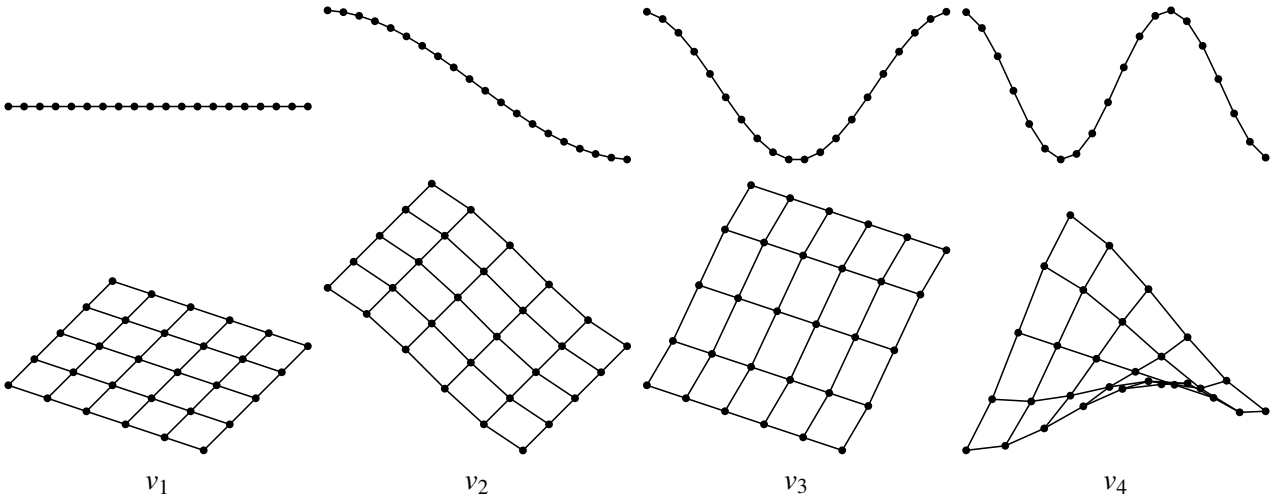

Figure 8: The first four eigenvectors of a path graph and a grid graph.

object whose vibrations are tiny displacements of the metal in the third dimension, perpendicular to the cymbal. Represent these displacements as a function $\phi(p)$ that maps each point p in the cymbal to its amplitude of vibration. A cymbal has *normal modes*, functions that represent the amplitude of motion of points vibrating with a shared frequency, which is called a *resonant frequency*. These modes represent patterns of motion in which the cymbal naturally "likes" to vibrate; vibrations in those modes last a long time. These modes are *eigenfunctions* (the continuous analogue of eigenvectors) of Laplace's equation $\nabla^2\phi = 0$, and have associated eigenvalues that are proportional to the resonant frequencies of the vibration (and of the sounds you hear).

The low-frequency modes of vibration appear as long waves, and the high-frequency modes as short waves. The long wavelength of the lowest-frequency mode (excluding the zero-frequency translational mode) is what makes it suitable for partitioning the cymbal: it divides the cymbal into two roughly balanced halves along the curve where the amplitude of vibration is zero. (It also determines the cymbal's lowest and loudest pitch.)

A computer can simulate a vibrating cymbal by modeling it discretely as a finite difference grid or a finite element mesh, either of which is essentially a weighted graph. It is no accident that the matrix L is called a Laplacian: the *stiffness matrices* used in finite difference or finite element methods to approximate a solution of Laplace's equation are Laplacian matrices. (In particular, the components of each row and column sum to zero.) The least eigenvalues of the stiffness matrix are good approximations to the least eigenvalues of Laplace's equation for the physical cymbal, and the corresponding discrete eigenvectors approximate the continuous eigenfunctions of the physical system.

2.6 Unbalanced Cuts: The Laplacian Objective Function, Revisited

When we relax the constraint that every component of x be ± 1, we create an apparent flaw: the balance constraint (5) appears to be impotent, because all it requires is that the components of x sum to zero. Suppose that G_1 has nine times as many vertices as G_2. If we set $x_i = 1/3$ for each vertex i in G_1, and $x_i = -3$ for each vertex in G_2, the balance constraint is satisfied, but the subgraphs aren't balanced.

This flaw is really a virtue. We will see that the relaxed optimization problem is a relaxation of not

merely one discrete optimization problem, but of multiple discrete optimization problems, each one assigning different masses to the subgraphs.

Observe that if the indicator vector x uses values other than ± 1 to denote G_1 and G_2, the objective function $x^T L x$ is no longer equal to $4\,\mathrm{Cut}(G_1, G_2)$. By Equation (3), $x^T L x = (c_1 - c_2)^2\,\mathrm{Cut}(G_1, G_2)$. The penalty factor $(c_1 - c_2)^2$ helps to fight excessive imbalance by preventing the components of x from getting too large in magnitude. Let's flesh out this insight for the more general partitioning problem with vertex masses.

Dhillon [12] observes that the relaxed optimization problem (13) is a relaxation of other discrete optimization problems besides the bisection problem we originally considered. Suppose the components of x are restricted to take on one of two values, c_1 or c_2, but those values are not necessarily ± 1. What values of c_1 and c_2 satisfy both constraints of problem (13)? The balance constraint $\mathbf{1}^T M x = 0$ can be rewritten $c_1\,\mathrm{Mass}(G_1) + c_2\,\mathrm{Mass}(G_2) = 0$, and the quadratic constraint $x^T M x = \mathrm{Mass}(G)$ can be rewritten $c_1^2\,\mathrm{Mass}(G_1) + c_2^2\,\mathrm{Mass}(G_2) = \mathrm{Mass}(G_1) + \mathrm{Mass}(G_2)$. Both constraints are satisfied by

$$c_1 = \sqrt{\frac{\mathrm{Mass}(G_2)}{\mathrm{Mass}(G_1)}}, \qquad c_2 = -\sqrt{\frac{\mathrm{Mass}(G_1)}{\mathrm{Mass}(G_2)}}. \tag{16}$$

So we see that the optimization problem (13) is a relaxation of the discrete optimization problem

$$\begin{aligned} &\text{minimize} && x^T L x \\ &\text{subject to} && \forall i,\, x_i = \sqrt{\frac{\mathrm{Mass}(G_2)}{\mathrm{Mass}(G_1)}} \text{ or } x_i = -\sqrt{\frac{\mathrm{Mass}(G_1)}{\mathrm{Mass}(G_2)}} \\ &\text{and} && \mathbf{1}^T M x = 0. \end{aligned} \tag{17}$$

This is true whether or not the masses of the subgraphs are fixed in advance. If we wish, we can make $\mathrm{Mass}(G_1)$ be an additional variable to optimize. The sweep cut described in Section 2.2 does exactly that.

Recalling Equation (3), we are minimizing

$$\begin{aligned} x^T L x &= (c_1 - c_2)^2\,\mathrm{Cut}(G_1, G_2) \\ &= \mathrm{Mass}(G)^2 \frac{\mathrm{Cut}(G_1, G_2)}{\mathrm{Mass}(G_1)\,\mathrm{Mass}(G_2)} && (18) \\ &= \mathrm{Mass}(G)\left(\frac{1}{\mathrm{Mass}(G_1)} + \frac{1}{\mathrm{Mass}(G_2)}\right)\mathrm{Cut}(G_1, G_2). && (19) \end{aligned}$$

Ignore the factor $\mathrm{Mass}(G)$, which is constant. Observe that this objective function is essentially the sparsity (or the equivalent average cut criterion) defined in Section 1.2. If the mass of a subgraph approaches zero, the objective function approaches infinity, thereby penalizing severely unbalanced partitions. This is good news, and it entails no change to the spectral partitioner.

Don't be misled, though: in practice, the sweep cut is usually effective for finding a cut with a good isoperimetric ratio—or for any other criterion that simplifies to minimizing the cut weight if we fix the subgraph masses as constants. The sweep cut investigates every promising value of $\mathrm{Mass}(G_1)$ (every possible value if the vertices all have equal mass), and for each one solves a relaxed optimization problem that minimizes the sparsity (18)—and thus the cut weight—subject to fixed subgraph masses. The magic is that every choice of $\mathrm{Mass}(G_1)$ yields the same relaxed problem (13)! The continuous optimization problem is a relaxation of many discrete problems, each balancing the subgraphs differently. The Fiedler vector v_2 encodes an approximate solution for all of them.

If the sweep cut can approximately optimize several different objective functions, what is special about the objective function (18)? The Fiedler vector minimizes (18) on the continuous domain, so if it miraculously has components with only two values, it encodes the discrete sparsest cut. Section F.4 describes a method that sometimes makes this miracle happen, thereby verifying that a cut optimizes the sparsity. If the Fiedler vector is not two-valued, it nevertheless tells us a lower bound on the objective function achieved by the discrete optimum. This interpretation of the objective function implies several bounds on the sparsity of the sweep cut and of the sparsest cut, discussed in Sections 2.7 and B.

2.7 Bounds on the Weight of an Unbalanced Cut

It is an obvious principle of constrained optimization that the minimum value $x^{\mathsf{T}}Lx$ obtained by the solution x to the relaxed optimization problem (13) is a lower bound for the minimum value obtained by the solution x to the more strongly constrained discrete optimization problem (17). Therefore, the "sparsity" of the relaxed solution is a lower bound for the sparsity of the unknown discrete solution. Likewise, the maximum "sparsity" for the relaxed problem is an upper bound for the sparsity of every discrete cut.

Section 1.5 gives inequalities bounding the weight of a bisection. Equation (18) enables us to generalize them to unbalanced cuts. Substituting (18) into (14) yields the inequalities

$$\frac{\lambda_2}{\text{Mass}(G)} \leq \frac{\text{Cut}(G_1, G_2)}{\text{Mass}(G_1)\,\text{Mass}(G_2)} \leq \frac{\lambda_n}{\text{Mass}(G)}, \tag{20}$$

which bound the sparsity of *every* cut, balanced or not, except the trivial empty cut. If we constrain $\text{Mass}(G_1)$ and $\text{Mass}(G_2)$ to be equal, we recover the bounds (15). But the bounds (20) apply to all cuts, not just bisections, so they imply some general conclusions about graph cuts. Every cut—in particular, the maximum cut—is subject to the upper bound

$$\text{Cut}(G_1, G_2) \leq \lambda_n \frac{\text{Mass}(G)}{4},$$

and that value can be obtained only by a balanced cut. This bound was proven by Mohar and Poljak [53].

Every cut has the lower bound

$$\text{Cut}(G_1, G_2) \geq \lambda_2 \frac{\text{Mass}(G_1)\,\text{Mass}(G_2)}{\text{Mass}(G)}. \tag{21}$$

If every vertex has equal mass, this bound implies that the weight of the minimum cut is at least $\lambda_2(n-1)/n$. This bound is the weighted-graph analogue of Fiedler's bound [22] on the edge connectivity of an unweighted G: Recall from Section 1.5 that if every edge has weight 1, every cut except the trivial cut has at least λ_2 edges. Fiedler's bound does not hold for weighted graphs; the weaker bound of $\lambda_2(n-1)/n$ is tight.

One practical use of the lower bound is as an indication of how close a rounded solution might be to optimality. When the spectral partitioning algorithm runs, it computes as a byproduct both lower and upper bounds on the sparsity of the optimal solution. The lower bound is $\lambda_2/\text{Mass}(G)$, which is the value achieved by the optimal solution $\sqrt{\text{Mass}(G)}\,v_2$ of the continuous optimization problem. The upper bound is the sparsity of the rounded solution returned by the partitioner. If the lower and upper bounds are close to each other, then the rounded solution must be close to the optimal solution. If not, Section F describes two methods that might find a better partition or improve the lower bound.

This discussion merely introduces the topic of spectral bounds on the cut weight. Section B derives *Cheeger's inequality*, which guarantees that if λ_2 is small, there exists a cut with a low isoperimetric ratio and low sparsity. Moreover, the sweep cut on the Fiedler vector is such a cut!

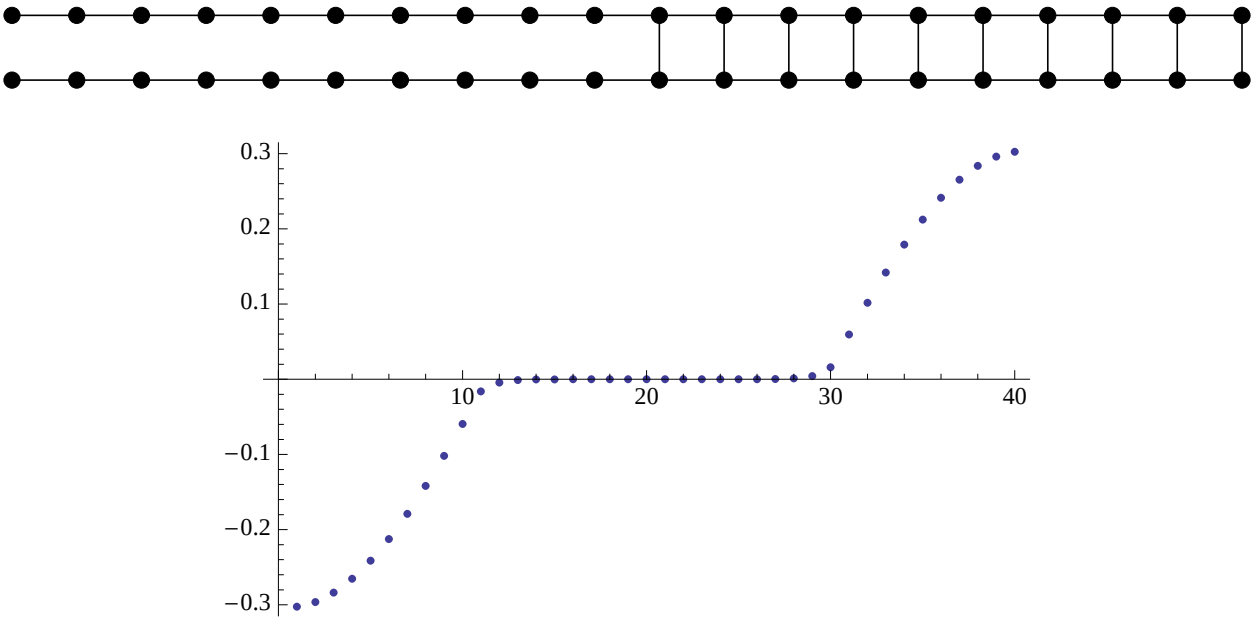

Figure 9: The nemesis of spectral bisection, the roach graph. Below the graph is a plot of its Fiedler vector, with the vertices plotted in clockwise order from antenna tip to antenna tip.

2.8 How Good Is Spectral Partitioning?

How well does the rounded result of the continuous optimization problem approximate the solution of the discrete optimization problem? Consider bisections first, then unbalanced cuts.

Let us begin with what is known about the difficulty of approximating a minimum bisection. Assume every edge weight and vertex mass is 1. Garey, Johnson, and Stockmeyer [25] show that the problem of finding a minimum bisection is NP-complete. The best polynomial-time approximation algorithm known is an $O(\log^{1.5} n)$-approximation algorithm; that is, it finds a bisection that cuts at most $O(e \log^{1.5} n)$ edges, where e is the number of edges in the minimum bisection. The algorithm combines a bisection algorithm of Feige and Krauthgamer [20] with a subsequent algorithm of Arora, Rao, and Vazirani [5] used as a subroutine.

There is no evidence that a polynomial-time $O(1)$-approximation algorithm for the minimum bisection cannot exist, but there are some inapproximability results. A *polynomial-time approximation scheme* is a $(1 + \epsilon)$-approximation algorithm that runs in polynomial time for any fixed $\epsilon > 0$, with the proviso that the running time may increase arbitrarily rapidly as ϵ approaches zero. Khot [40] shows that there is no polynomial-time approximation scheme for graph bisection unless NP-complete problems can be solved in randomized subexponential time. (Complexity theorists doubt that they can be, but it is not as unlikely as P = NP.) Bui and Jones [8] give an inapproximability result that holds not only for exact bisections, but also for bipartitions where the smaller subgraph must have at least some fixed proportion (e.g. 1%) of the vertices. They show that for any $\epsilon > 0$, it is NP-hard to find an acceptably balanced cut of at most $e + n^{2-\epsilon}$ edges, where e is the number of edges in the minimum acceptably balanced cut. Graphs with a small maximum vertex degree cannot have many edges, but even for graphs with maximum degree 3, it is NP-hard to find an acceptably balanced cut of at most $e + n^{1/2-\epsilon}$ edges.

As an approximation algorithm, spectral partitioning with the median cut sometimes fails spectacularly. The best known example is the *roach graph* of Guattery and Miller [30], illustrated in Figure 9. The roach

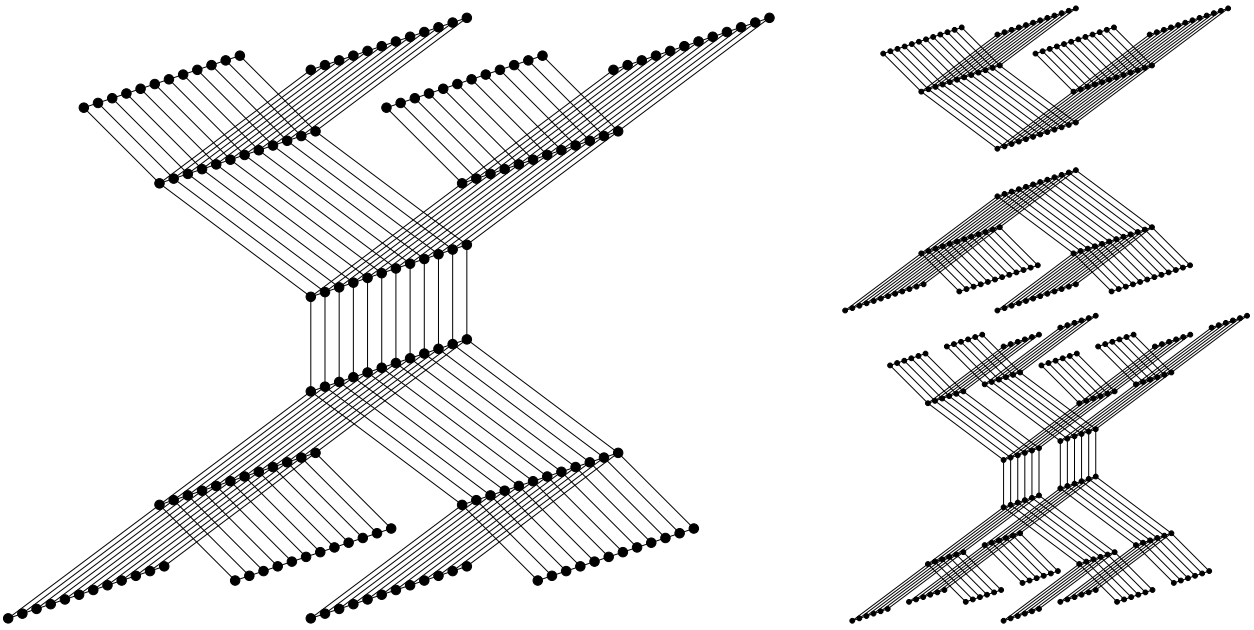

Figure 10: The cross product of a path and a double tree (left), its sparsest cut (upper right), and its spectral sweep cut (lower right).

graph looks like a ladder in which exactly half the rungs are missing, giving it the appearance of a cockroach with two long antennae. The optimal bisection cuts just two edges, severing the antennae from the body. Guattery and Miller show that the median cut cleaves the roach lengthwise, cutting $n/4$ edges.

However, spectral bisection's theoretical reputation has been partly salvaged for some classes of graph. It is necessary to replace the median cut with another bisection method built from multiple sweep cuts, described in Section B.2. This method is guaranteed to find a bisection of any planar graph cutting at most $8d\sqrt{n}$ edges, where d is the maximum vertex degree. (See Section B for some similar results.) For the roach graph, it does better: it finds the optimal cut. This method computes eigenvectors of multiple Laplacian matrices, so it is slower than the median cut.

Unbalanced cuts seem to be easier to approximate than minimum bisections. Again, assume every edge weight and vertex mass is 1. Consider approximating the cut with the minimum isoperimetric ratio $\text{Cut}(G_1, G_2)/\min\{|\mathcal{V}_1|, |\mathcal{V}_2|\}$ or the minimum uniform sparsity $\text{Cut}(G_1, G_2)/(|\mathcal{V}_1|\,|\mathcal{V}_2|)$. (The two criteria are roughly equivalent, differing by a factor between $n/2$ and n.) Mohar [52] shows that it is NP-hard to determine the minimum isoperimetric ratio, and Matula and Shahrokhi [50] show that it is NP-hard to find the uniform sparsest cut. The best polynomial-time approximation algorithm known, by Arora, Rao, and Vazirani [5], finds an $O(\sqrt{\log n})$-approximation to the uniform sparsest cut or the cut of minimum isoperimetric ratio.

Ambühl, Mastrolilli, and Svensson [3] extend Khot's inapproximability result for bisections and show that there is no polynomial-time approximation scheme for the sparsest cut unless NP-complete problems can be solved in randomized subexponential time. It is a crucial open problem whether a polynomial-time $O(1)$-approximation algorithm for the uniform sparsest cut exists.

The spectral sweep cut is better at finding a sparse cut than the spectral median cut is at finding a good bisection, but Guattery and Miller [30] show that the sweep cut can miss the optimum by a factor of $\Theta(n^{1/3})$. Consider the graph in Figure 10, which is the *cross product graph* of a path and a *double tree*, the latter

consisting of two complete balanced binary trees whose roots are connected by an edge. (See Guattery and Miller for formal definitions of these terms. Most readers will get the idea much more quickly from the figure than from the definitions.) If the double tree has $p \geq 14$ vertices and the number of path vertices is just over $\pi\sqrt{p}$, then the sparsest cut is the bisection that separates each double tree into two trees, cutting $\Theta(\sqrt{p}) = \Theta(n^{1/3})$ edges and thus having an isoperimetric ratio in $\Theta(1/n^{2/3})$. But Guattery and Miller show that the sweep cut is the bisection that separates two copies of the double tree, cutting $p \in \Theta(n^{2/3})$ edges and thus having an isoperimetric ratio in $\Theta(1/n^{1/3})$.

The good news is that the sweep cut cannot do worse. Spectral partitioning is an approximation algorithm in the following sense: if a graph has an isoperimetric number (minimum isoperimetric ratio) of h, the sweep cut has an isoperimetric ratio less than $2\sqrt{dh}$, where d is the maximum vertex degree. Thus spectral partitioning can be characterized as a $2\sqrt{d/h}$-approximation algorithm or a $\sqrt{8d/\lambda_2}$-approximation algorithm. The main tool for proving this and related bounds on the quality of a sweep cut is *Cheeger's inequality*. See Section B for a proof of Cheeger's inequality, a generalization of this approximation result to arbitrary edge weights and vertex masses, and other implications of the inequality.

One other implication of Cheeger's inequality is that the sweep cut works well on some classes of graphs that are partitioned frequently in practice, such as bounded-degree planar graphs and finite element meshes. For any bounded-degree planar graph, the sweep cut has an isoperimetric ratio in $O(1/\sqrt{n})$. For any bounded-degree high-quality tetrahedral mesh, the sweep cut has an isoperimetric ratio in $O(1/n^{1/3})$. See Section B.1 for details. These isoperimetric ratios are asymptotically optimal for these classes of graphs, though of course some graphs in these classes have much smaller cuts. Unfortunately, graphs having vertices with very large degree may have no small cut. For instance, if a vertex is connected to every other vertex, then every bisection cuts more than half the edges adjoining it. (However, every planar graph has a small vertex separator; see Section C.)

Spectral partitioning works well in most practical applications, and even the median cut is usually a good bisection [58, 62]. But spectral partitions are rarely optimal; they usually can be improved by local optimization methods. The most famous is the Kernighan–Lin heuristic [38] and its faster version by Fiduccia and Mattheyses [21]. Spectral partitioning tends to find cuts that look good globally (if you squint), but flawed locally. The Kernighan–Lin algorithm does the opposite. A combination of the two, wherein Kernighan–Lin cleans up a spectral partition, offers the best of both [58].

3 Isoperimetric Graph Partitioning

3.1 Graph Partitioning as Constrained Quadratic Optimization, Revisited

The main pitfall of spectral graph partitioning is that algorithms for extracting the Fiedler vector v_2 (see Section H) are slow and complicated. Grady and Schwartz [29] propose a different relaxation of the discrete optimization problem that is easier to solve, because it requires only the solution of a linear system rather than the computation of an eigenvector.

Their method requires a tiny change to the balance constraint. It also requires explicit enforcement of the balance constraint, whereas in spectral partitioning the eigenvectors enforce the balance constraint (12) for free, by virtue of $\mathbf{1}$ being an eigenvector with all the other eigenvectors M-orthogonal to it.

For technical reasons, the new balance constraint is

$$\mathbf{1}^\mathsf{T} M x = \mu, \tag{22}$$

where μ is any constant you wish to choose *except* zero. As in Figure 2, this constraint defines a cross-section of the cylindrical isosurfaces that is M-orthogonal to the central axis, but the cross-section no longer passes through the origin.

At first blush, you might think that the requirement that $\mu \neq 0$ means that we can request an unbalanced partition of the graph, but not an exact bisection. Fortunately, that appearance is false, because we can also choose the constants c_1 and c_2. Isoperimetric partitioning employs a *0-1 indicator vector* x, which means we choose $c_1 = 1$ to signify which vertices are in G_1 and $c_2 = 0$ to signify which are in G_2. The constraint (22) specifies that

$$\text{Mass}(G_1) = \mu.$$

If we want a bisection, we simply choose $\mu = \text{Mass}(G)/2$.

Thus, we reduce the graph partitioning problem to the discrete optimization problem of choosing x to

$$\begin{array}{rl} \text{minimize} & x^\mathsf{T} L x \\ \text{subject to} & \forall i, x_i = 0 \text{ or } x_i = 1 \\ \text{and} & \mathbf{1}^\mathsf{T} M x = \mu, \end{array} \tag{23}$$

where $\mu > 0$ is the desired mass of G_1.

3.2 The Relaxed Optimization Problem, Revisited

We would like to relax the optimization problem (23) by simply dropping the first constraint, but we get into trouble. The minimum value of $x^\mathsf{T} L x$ (namely zero) is attained by assigning every vertex the same value, as Equation (2) indicates. If we choose $x = (\mu/\mathbf{1}^\mathsf{T} M \mathbf{1})\mathbf{1}$, the balance constraint is satisfied, but we don't have a partition.

This failing is a consequence of the fact that L is singular. The relaxed problem is analogous to solving Laplace's equation with a finite difference method without setting a boundary condition. As we shall see in Section 3.5, it is also analogous to injecting current into an electrical circuit without attaching it to a ground.

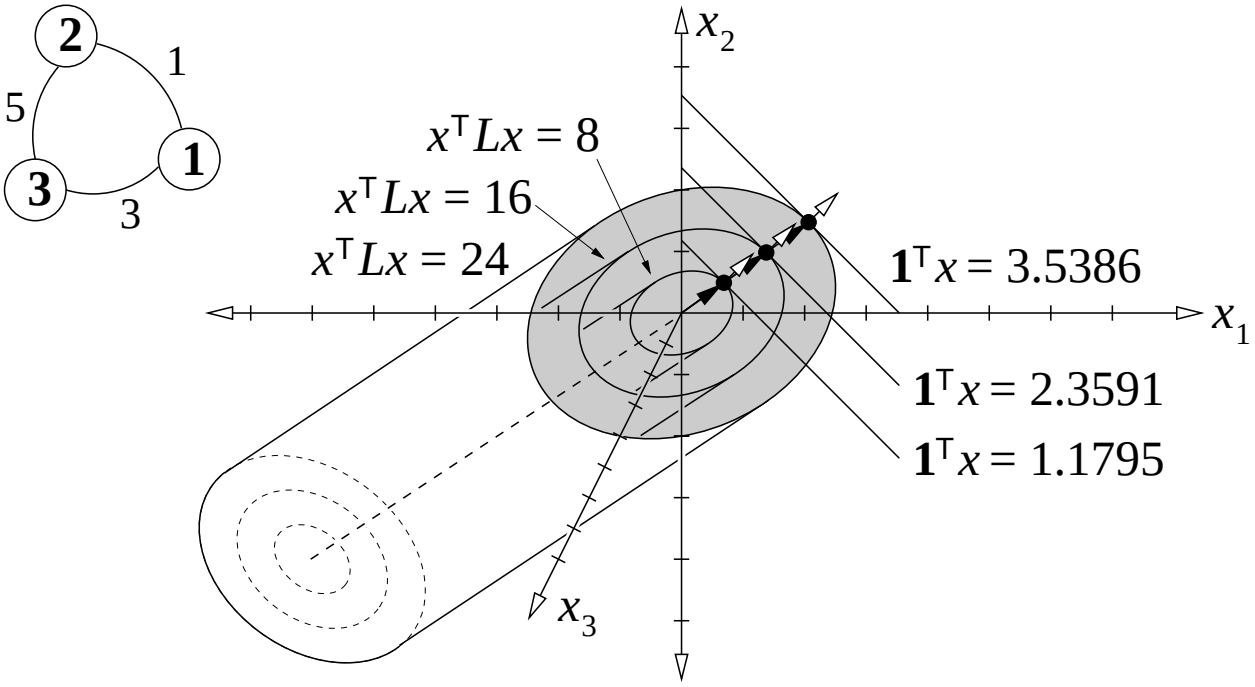

Figure 11: Isosurfaces of the function $x^\mathsf{T}Lx$ for the graph at upper left. The shaded cross-section represents the linear constraint $x_3 = 0$. The linear balance constraint $\mathbf{1}^\mathsf{T}x = \mu$ is drawn for several values of μ. Black-tipped vectors are solutions x for different values of μ, and white-tipped vectors are local gradients of the objective function. Observe that μ determines the magnitude but not the direction of the solution.

A solution is to constrain one vertex i to lie in G_2 by constraining x_i to be zero in the optimization problem (23). Call vertex i the *ground vertex*. By the symmetry of G_1 and G_2, this constraint does not rule out finding the optimal cut. Substituting $x_i = 0$ yields the problem of choosing $\hat{x}$ to

$$\begin{aligned} &\text{minimize} && \hat{x}^\mathsf{T}\hat{L}\hat{x} \\ &\text{subject to} && \mathbf{1}^\mathsf{T}\hat{M}\hat{x} = \mu, \end{aligned} \tag{24}$$

where $\hat{x}$ is x with the ith component removed, and $\hat{L}$ and $\hat{M}$ are produced by removing the ith row and ith column from L and M. If G is not connected, we must designate one ground vertex in each connected component of G. This ensures that $\hat{L}$ is nonsingular. (Section F.1 explains how to fix vertices at nonzero values, if desired.)

Figure 11 depicts the cross-section induced by the constraint $x_3 = 0$ in the cylinders from Figure 2. This cross-section spans the x_1-x_2 plane, and cuts the central axis of the cylinders diagonally. The constraint $\mathbf{1}^\mathsf{T}Mx = \mu$ also induces a cross-section (in the figure, $M = I$ and the constraint is $\mathbf{1}^\mathsf{T}x = \mu$), and the two cross-sections intersect in a line $\mathbf{1}^\mathsf{T}\hat{M}\hat{x} = \mu$. (If the graph had more than three vertices, the intersection would be an $(n-2)$-dimensional affine subspace.)

If μ is zero, $x^\mathsf{T}Lx$ is minimized by $x = \mathbf{0}$, which is why we forbid the choice $\mu = 0$.

3.3 Solving the Relaxed Optimization Problem

It is a well known property of constrained numerical optimization that a necessary condition for $\hat{x}$ to be a solution of the relaxed constrained optimization problem (24) is

$$\nabla(\hat{x}^\mathsf{T}\hat{L}\hat{x}) = \beta\,\nabla(\mathbf{1}^\mathsf{T}\hat{M}\hat{x}) \tag{25}$$

for some real-valued Lagrange multiplier β. To see why Equation (25) holds,[6] consider Figure 11. The black points in the figure minimize $\hat{x}^\mathsf{T}\hat{L}\hat{x}$ for different values of μ. It is visually apparent that the minima occur where a linear constraint ($\mathbf{1}^\mathsf{T}\hat{M}\hat{x} = \mu$) is tangent to an elliptical isosurface, because any motion of a minimum point along the linear constraint will move it outside its ellipse and increase the objective function. Therefore, the minima are points where the gradient of $\hat{x}^\mathsf{T}\hat{L}\hat{x}$ (white-tipped vectors) points in the same direction as a vector orthogonal to the linear constraint, which is what Equation (25) expresses.

Taking the gradients, we have[7]

$$2\hat{L}\hat{x} = \beta\hat{M}\mathbf{1}. \tag{26}$$

We solve the system (26) in two steps. First, solve the linear system

$$\hat{L}y = \hat{M}\mathbf{1} \tag{27}$$

for y. Second, choose β so that $\hat{x} = (\beta/2)y$ satisfies the balance constraint $\mathbf{1}^\mathsf{T}\hat{M}\hat{x} = \mu$; substituting that choice gives $\hat{x} = (\mu/\mathbf{1}\hat{M}y)y$.

It is fortunate that the first step is independent of the value of μ. As with spectral partitioning, only after doing most of the work (solving the linear system) do we need to decide how we want to balance the partition. (Figure 11 gives visual intuition for why the direction of $\hat{x}$ is independent of μ.)

This independence justifies the use of a sweep cut (as described in Section 2.2) to try to minimize the isoperimetric ratio, the sparsity, or the normalized cut criterion. Recall that if we fix the subgraph masses as constants, these three criteria all simplify to minimizing the cut weight. The solution vector y, illustrated in Figure 12, encodes a solution of the continuous optimization problem (24) for *every* choice of μ. By rounding y to a 0-1 indicator vector, we produce what we hope is an approximate solution of the discrete optimization problem (23) for a specific choice of $\mu = \text{Mass}(G_1)$. The sweep cut heuristic tries out $n - 1$ different roundings, each assigning a different mass to G_1, and returns the rounding that optimizes the chosen criterion. By attempting to approximate a minimum cut for every practical choice of $\text{Mass}(G_1)$, the heuristic finds a cut that tries to minimize the sparsity or isoperimetric ratio.

In the three-vertex graph in Figure 11, $\hat{x}$ is some multiple of $[7, 5]^\mathsf{T}$, so x is a multiple of $[7, 5, 0]^\mathsf{T}$. This solution could be rounded to separate vertex 1 from the others, giving the best possible cut, or to separate vertex 3 from the others, giving the worst possible cut. A sweep cut will choose the former. However, the ordering of the vertices implied by the indicator vector is not ideal. The spectral partitioner puts vertex 3 in the middle, as it is more tightly bound to the other two vertices than they are to each other. The isoperimetric partitioner precluded this outcome by fixing vertex 3 at zero. In graphs having more than three vertices, the choice of ground vertex influences the quality of the partition. Isoperimetric partitioning seems unlikely to find the optimal cut if the ground vertex is too close to the optimal cut.

[6] At the time of this writing, the Wikipedia page for "Lagrange multiplier" is a good resource for understanding Equation (25).

[7] If we had not constrained one of the vertices, the system (26) would be $2Lx = \beta M\mathbf{1}$, which is overconstrained and has no solution except the trivial $\beta = 0$, $x = \mathbf{0}$. To see this, recall that the components of any column of L sum to zero; therefore, so do the components of $2Lx$, whereas the components of $M\mathbf{1}$ are all positive.

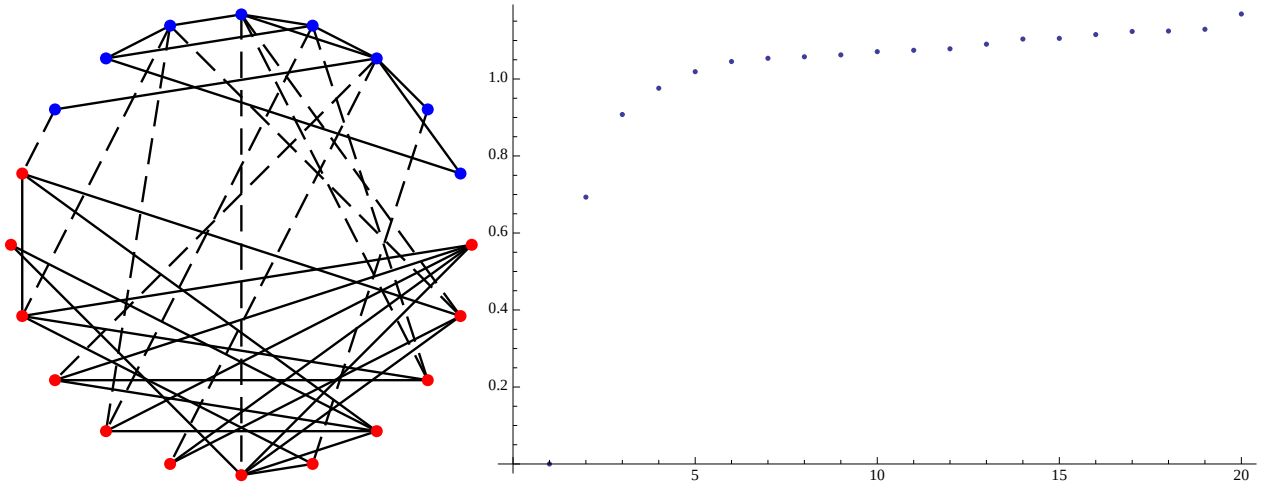

Figure 12: A graph and a corresponding isoperimetric solution vector y. All the edge weights and vertex masses are 1. The vertices have been sorted in clockwise order by increasing component in y. The graph is partitioned by a sweep cut that finds the threshold offering the minimum sparsity.

Grady and Schwartz [29] call their method *isoperimetric partitioning* because the continuous problem optimizes the isoperimetric ratio

$$\frac{\text{Cut}(G_1, G_2)}{\text{Mass}(G_1)} = \frac{x^\mathsf{T} L x}{\mathbf{1}^\mathsf{T} M x}, \tag{28}$$

subject to the constraint that its denominator is fixed to be μ. (This expression is the classical isoperimetric ratio only when $\text{Mass}(G_1) \leq \text{Mass}(G)/2$, but see below for an application where this asymmetric definition of the ratio makes more sense.)

The minimum value $x^\mathsf{T} L x$ obtained by the solution x to the relaxed optimization problem (24) is a lower bound for the minimum value obtained by the solution x to the more strongly constrained discrete optimization problem (23). (Recall that Section 2.7 describes a use for such a lower bound.) The same is true for spectral partitioning, but the objective function is optimized subject to different constraints. In isoperimetric partitioning, the "isoperimetric ratio" of the relaxed solution is a lower bound for the isoperimetric ratio of the discrete solution, but a similar statement cannot be made for the sparsity. In spectral partitioning, the "sparsity" of the relaxed solution is a lower bound for the sparsity of the discrete solution, but a similar statement cannot be made for the isoperimetric ratio. However, as we have already seen, isoperimetric partitioning with the sweep cut works reasonably well for finding a sparse cut, although it does not provide a lower bound on the sparsest cut.

The name "isoperimetric ratio" originates in geometry, where it refers to an object's ratio of perimeter to area, or its ratio of surface area to volume. Consider a partition of space into a finite number of cells, including one unbounded "outer cell." Suppose we want to find the subset of the bounded cells whose union minimizes the ratio of surface area to volume. Let G be a graph that has one vertex for each cell. Assign each vertex a mass equal to the corresponding cell's volume. If two cells share a common boundary, G connects them with an edge whose weight is the surface area of the shared boundary. For a 0-1 indicator vector x, $\mathbf{1}^\mathsf{T} M x$ is the volume assigned to G_1 and $x^\mathsf{T} L x$ is the surface area separating G_1 from G_2. We optimize the geometric isoperimetric ratio by optimizing its combinatorial analogue (28) with the unbounded outer cell constrained to be in G_2.

Recall that finding a cut that minimizes the isoperimetric ratio is NP-hard [52].

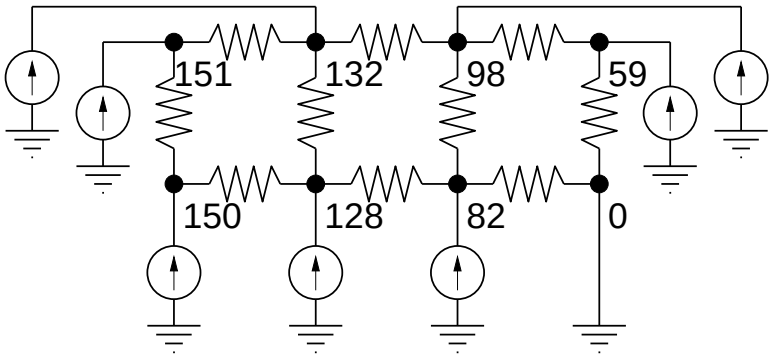

Figure 13: An isoperimetric partitioner's linear system models voltages in a resistor circuit with current sources. The number next to each vertex is its approximate voltage, given equal resistors and equal current sources.

3.4 The Isoperimetric Partitioning Algorithm

Given a weighted graph G, the vertex masses, and the desired mass μ of G_1, construct G's Laplacian matrix L and the mass matrix M. Choose a ground vertex in each connected component of G and fix its value to zero so it lies in G_2. Define $\hat{L}$ and $\hat{M}$ by deleting the ground vertices' rows and columns from L and M.

Solve the linear system $\hat{L}y = \hat{M}\mathbf{1}$ for y, and optionally compute $\hat{x} = (\mu/\mathbf{1}\hat{M}y)y$.

Round y or $\hat{x}$ as described in Section 2.2. We need $\hat{x}$ only if we want to round with a preselected nonzero threshold, which people rarely do in practice. The other rounding methods from Section 2.2 depend only on the order of the sorted components of the vector, and not on their precise values, so they can operate on y.

The solution y does not need to be accurate to many decimal places, as reassigning or swapping components near the threshold is almost as likely to improve the cut as to make it worse. This makes iterative methods for solving the linear system (27) attractive, especially the conjugate gradient method [35], which can often produce a good-enough solution faster than a direct method.

3.5 The Circuit Analogy

The linear system (27) models an electric circuit, illustrated in Figure 13, and the connection provides intuition about how isoperimetric partitioning works.

Suppose that each edge with weight w_{ij} represents a resistor with conductivity w_{ij} (that is, resistance $1/w_{ij}$) connecting vertex i to vertex j. Each mass m_i represents a current source that injects m_i amperes of current into vertex i. Current leaves the circuit through the ground vertex, which is fixed at voltage zero.

By the following argument, the solution y indicates the potential (voltage) at each vertex of this circuit. By Ohm's law, the current flowing from vertex i to vertex j is $(y_i - y_j)w_{ij}$. Kirchhoff's current law states that the sum of currents flowing out of a vertex is zero, so for each vertex i,

$$\sum_{j:(i,j)\in\mathcal{E}} (w_{ij}y_i - w_{ij}y_j) = m_i,$$

which is the ith row of the system (27).

The circuit analogy reveals several properties of isoperimetric partitioning.

- The reason we need to fix a vertex is that if we connect current sources to a circuit but provide no ground vertex for current to exit through, current cannot flow, and there is no solution.
- Because all the current sources are positive and there is no negative voltage source, every vertex must have a nonnegative voltage; hence, the solution y of (27) has no negative component.
- If there is only one ground vertex, and we choose a threshold cut—that is, G_2 contains the vertices i for which x_i is less than some threshold—then G_2 is connected, because every vertex in G_2 has a path of monotonically decreasing voltage to the ground vertex.

These statements can be proven formally [29], but the circuit analogy is more revealing than the proofs.

3.6 Spectral vs. Isoperimetric Partitioning

Spectral and isoperimetric partitioning minimize the same objective function, subject to balance constraints that are only superficially different. The real difference between spectral and isoperimetric partitioning is in the other constraints. The spectral partitioner optimizes $x^T Lx$ subject to the quadratic constraint that $x^T Mx$ is fixed. The isoperimetric partitioner optimizes $x^T Lx$ subject to the constraint that a vertex value is fixed. The former constraint treats the vertices more symmetrically. The latter constraint tends to yield a numerical problem that is easier to solve.

Spectral partitioning is an approximation algorithm with a rich theory governing the relationship between the sparsest cut and the smallest eigenvalues and their eigenvectors. Isoperimetric partitioning does not appear to lend itself to an analogue of Cheeger's inequality (Section B), which establishes spectral partitioning's claim to be an approximation algorithm.

In their experiments, Grady and Schwartz [29] find that isoperimetric partitioning runs three to ten times faster than spectral partitioning, and produces results of similar quality (with spectral partitioning winning some contests, and isoperimetric partitioning winning others). The spectral partitioner's symmetric treatment of the vertices sometimes helps the quality of the partitions, but it can be a liability when a graph has symmetries that must be broken arbitrarily. Figure 14 shows a (somewhat contrived) example of a graph whose symmetry works against spectral bisection and in favor of isoperimetric bisection. Isoperimetric partitioning is sensitive to the choice of the ground vertex, which naturally breaks symmetries and sometimes improves performance on nearly symmetric graphs, but can compromise the quality of the partition if the chosen vertex is close to the true minimum cut. Isoperimetric partitioning is a new algorithm, and it is too soon to know how it performs across a variety of applications.

The running times of spectral and isoperimetric partitioning both depend on the spectrum of the Laplacian matrix L (or the normalized Laplacian $M^{-1/2}LM^{-1/2}$), albeit in different ways. Most implementations of spectral partitioning for large graphs find the Fiedler vector v_2 iteratively, with help from the Lanczos algorithm [43]. The speed with which the Lanczos algorithm can determine λ_2 (a prerequisite to computing v_2; see Section H) depends in part on the separation between the second and third eigenvalues, λ_2 and λ_3. If they are close together—as is common for graphs with near-symmetries—convergence may slow appreciably. One reason can be inferred from Figure 3. If $\lambda_2 \approx \lambda_3$, the elliptical isosurfaces in that diagram become nearly circular. The quadratic constraint $x^T x = n$ is also circular, so it becomes difficult to precisely locate the point where the constraint kisses the innermost isosurface. Performance is best when λ_2 is well-separated from λ_3. (However, if $\lambda_2 = \lambda_3$, then it is the separation between λ_3 and λ_4 that matters, and if the difference between λ_2 and λ_3 is tiny, we can sometimes stop the iterations early, not knowing if we have computed v_2 or v_3 or some linear combination of them.)

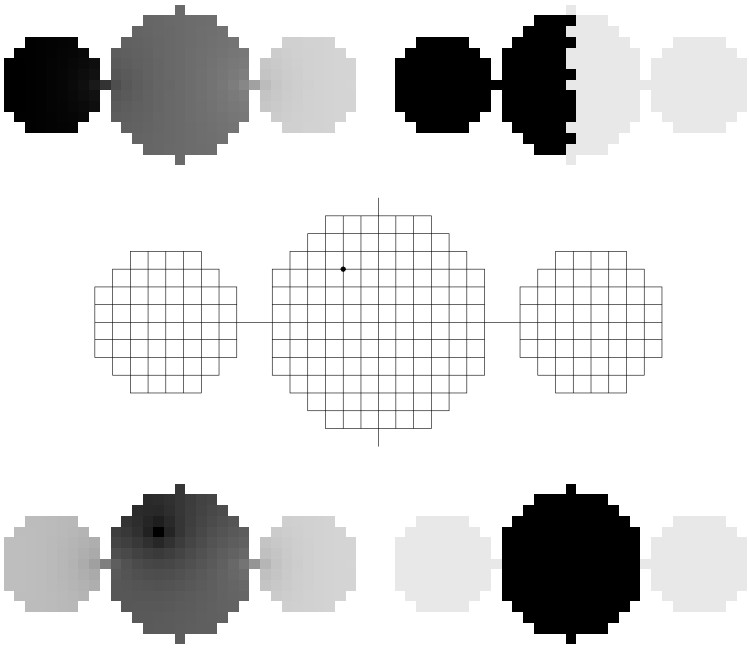

Figure 14: Spectral and isoperimetric median cuts on the graph at center. The Fiedler vector appears at upper left, and the spectral bisection at upper right. The isoperimetric solution vector appears at lower left (for the ground vertex indicated in the center illustration), and the isoperimetric bisection at lower right. Courtesy of Leo Grady and Eric Schwartz.

The linear system (27) is most commonly solved iteratively with the conjugate gradient algorithm [35], which is almost entirely insensitive to the value of λ_3 or to graph symmetry. Its speed depends in part on the *condition number* of $\hat{L}$ (or $\hat{M}^{-1/2}\hat{L}\hat{M}^{-1/2}$); it is fastest when the condition number is small. The condition number tends to grow with the size of the graph, the ratio between the largest and smallest vertex masses, and the ratio between the maximum and minimum cut weights. However, the condition number of $\hat{L}$ is not an entirely reliable guide; if the eigenvalues of $\hat{L}$ are mostly tightly clustered with a few outliers, the conjugate gradient method will perform almost as well as if the outliers were absent. For example, we can inflate the largest eigenvalue of $\hat{L}$ by attaching one new vertex of very small mass to the rest of the graph via one new edge of very large weight, but one extra iteration of the conjugate gradient method suffices to nullify its effect.

The Lanczos algorithm is difficult to implement robustly; see Section H for some reasons. The conjugate gradient algorithm is much easier.

Part II
Variations on a Theme

The remainder of this report describes variations and implications of the spectral and isoperimetric partitioning algorithms. There are sections on the normalized cut criterion used in image segmentation and random walks on graphs (Section A); vertex separators, which cut vertices rather than edges (C); using multiple eigenvectors or isoperimetric solutions to find better cuts (D); partitioning a graph into more than two subgraphs (E); constraining vertex values or otherwise biasing the partition (F); negative edge weights (G); computing eigenvectors (H); and bipartite graphs (I), which are sometimes more efficiently treated by the singular value decomposition. Most importantly, Section B explores Cheeger's inequality governing the isoperimetric number of a graph, with a proof that the spectral sweep cut achieves the bound.

The sections are almost independent of each other and may be read selectively. Most of the topics here are less settled than the topics in Part I, and are rich with potential for future research.

A The Normalized Cut

In the study of random walks on graphs, it is natural to choose the vertex masses to be the diagonal components of L; that is, $M_{ii} = m_i = L_{ii} = \sum_{j:(i,j)\in\mathcal{E}} w_{ij}$. Shi and Malik [61] popularized this choice of masses for problems in image segmentation, and named the cut thus derived the *normalized cut* (because the generalized Laplacian $M^{-1/2}LM^{-1/2}$, having 1's on the diagonal, is also known as the *normalized Laplacian*). A rationale comes from the following observation. Let $\text{Weight}(G_1)$ be the total weight of the edges in G_1—edges that are not cut. Define $\text{Weight}(G_2)$ likewise. As every edge contributes mass to each of its two vertices,

$$\text{Mass}(G_1) = 2\,\text{Weight}(G_1) + \text{Cut}(G_1, G_2) \quad \text{and} \quad \text{Mass}(G_2) = 2\,\text{Weight}(G_2) + \text{Cut}(G_1, G_2).$$

We can rewrite the objective function (19) as

$$\begin{aligned} x^{\mathsf{T}}Lx &= \text{Mass}(G)\left(\frac{\text{Cut}(G_1,G_2)}{\text{Mass}(G_1)} + \frac{\text{Cut}(G_1,G_2)}{\text{Mass}(G_2)}\right) \\ &= 2\,\text{Mass}(G)\left(1 - \frac{\text{Weight}(G_1)}{\text{Mass}(G_1)} - \frac{\text{Weight}(G_2)}{\text{Mass}(G_2)}\right). \end{aligned}$$

Thus, minimizing $x^{\mathsf{T}}Lx$ is equivalent to maximizing what Shi and Malik call the *normalized association*,

$$\frac{\text{Weight}(G_1)}{\text{Mass}(G_1)} + \frac{\text{Weight}(G_2)}{\text{Mass}(G_2)}.$$

Each fraction is the ratio of a subgraph's internal connectivity to its total internal and external connectivity.

When every vertex has a mass of 1, the sparsest cut rewards cuts that are balanced so each subgraph has roughly half the vertices, whereas the normalized cut objective function favors cuts balanced so each subgraph has roughly half the internal connectivity (uncut edge weight). A cut that balances the vertices poorly may be preferred if the smaller subgraph has strong internal connectivity. Shi and Malik observe that the normalized cut works better for image segmentation than the sparsest cut with equal masses.

In an appendix to Shi and Malik's paper, Christos Papadimitriou shows that finding the optimal discrete normalized cut is NP-hard, even for planar graphs.

B Cheeger's Inequality and a Guarantee for the Sweep Cut

Fiedler [22] calls the second eigenvalue λ_2 the *algebraic connectivity* of G—not only because G is connected if and only if $\lambda_2 > 0$, but also because the magnitude of λ_2 is a measure of how strongly the graph is connected. This relationship is quantified by *Cheeger's inequality*, a famous bound on a graph's isoperimetric number (the minimum isoperimetric ratio among all possible cuts), namely

$$\frac{\lambda_2}{2} \leq \min_{G_1} \frac{\text{Cut}(G_1, G_2)}{\min\{\text{Mass}(G_1), \text{Mass}(G_2)\}} \leq \sqrt{2\lambda_2 \max_{1 \leq i \leq n} \frac{L_{ii}}{M_{ii}}}, \tag{29}$$

where λ_2 is the second-smallest eigenvalue of the generalized eigensystem $Lv = \lambda Mv$. The left inequality follows from Inequality (20). The right inequality is the real Cheeger's inequality (though sometimes both are called Cheeger's inequality), and its proof appears in Section B.3.

The right inequality is valuable in at least two ways. First, it helps to bound the running times of approximate counting algorithms [63]. Second, its proof is constructive: the proof demonstrates that the sweep cut (with the minimum isoperimetric ratio criterion) on the Fiedler vector always satisfies the inequality. Therefore, it guarantees that the spectral partitioning algorithm can find a cut with isoperimetric ratio no greater than $\sqrt{2\lambda_2 \max_i(L_{ii}/M_{ii})}$.

Because both the optimum cut and the sweep cut satisfy the inequalities (29), the two inequalities together guarantee that the spectral partitioner can find a cut with isoperimetric ratio at most $2\sqrt{h \max_i(L_{ii}/M_{ii})}$, where h is the isoperimetric number of G. On this result rests spectral partitioning's claim to be an approximation algorithm, overestimating h by at most a factor of $2\sqrt{\max_i(L_{ii}/M_{ii})/h} \leq \sqrt{8 \max_i(L_{ii}/M_{ii})/\lambda_2}$.

In the literature, the right inequality usually appears in a simpler (but less general) form. Recall from Section A that in the transition graphs used to study random walks, and in the normalized cut criterion, $M_{ii} = L_{ii}$. For this choice, the upper bound is simply $\sqrt{2\lambda_2}$, and the spectral partitioner overestimates h by at most a factor of $2/\sqrt{h}$ (here $h \leq 1$). Alternatively, if every edge weight and every vertex mass is 1, then $\max_i(L_{ii}/M_{ii})$ is the maximum vertex degree d, and the upper bound is $\sqrt{2d\lambda_2}$. For this special case only (and only if G is not a complete graph with three or fewer vertices), Mohar [52] proves that a stronger version of the right inequality holds: the upper bound is at most $\sqrt{\lambda_2(2d - \lambda_2)}$.

Both inequalities, but especially the left, help to bound the mixing times of random walks on a graph G. The cut with the smallest isoperimetric ratio is a bottleneck that throttles the rate of mixing. If λ_2 is sufficiently large, the left inequality guarantees that random walks on that graph mix quickly. In this context, λ_2 is often called the *spectral gap* or *eigenvalue gap* (referring to the gap between $\lambda_1 = 0$ and λ_2).

A pair of bounds on the sparsest cut are sometimes also called Cheeger's inequality:

$$\frac{\lambda_2}{\text{Mass}(G)} \leq \min_{G_1} \frac{\text{Cut}(G_1, G_2)}{\text{Mass}(G_1)\,\text{Mass}(G_2)} \leq \frac{1}{\text{Mass}(G)} \sqrt{8\lambda_2 \max_{1 \leq i \leq n} \frac{L_{ii}}{M_{ii}}}.$$

The left sparsity inequality is taken from Inequality (20), and is a stronger bound than the left isoperimetric inequality (29). It dates back at least to Mohar [52]. The right sparsity inequality follows directly from (and is weaker than) the right isoperimetric inequality (29).

Cheeger's inequality was proven independently by Sinclair and Jerrum [63], Lawler and Sokal [45], and Mohar [52], building on earlier, similar results by Dodziuk [16] and Alon and Milman [1]. It is named after a theorem in geometry by Jeff Cheeger [9] that relates an eigenvalue of the Laplace–Beltrami operator on a Riemannian manifold to the manifold's isoperimetric number. Cheeger's original theorem is a continuous function-space analogue of the right inequality (29).

graph	λ_2	isoperimetric ratio	bisection edges
path	$4\sin^2(\pi/2n) < (\pi/n)^2$	$2/n$ or $2/(n-1)$	1
cycle	$4\sin^2(\pi/n) < (2\pi/n)^2$	$4/n$ or $4/(n-1)$	2
planar	$\leq 8d/n$	$\leq 4d/\sqrt{n}$	$\leq 8d(\sqrt{n}-1)$
high-quality tetrahedral mesh	$O(d/n^{2/3})$	$O(d/n^{1/3})$	$O(dn^{2/3})$
maximum vertex degree $d < n-1$	$\leq d$	$\leq d$	$\leq d(n-1)$
expander	$\Omega(1)$	$\Omega(1)$	$\Omega(n)$
complete graph	n	$\lceil n/2 \rceil$	$\lfloor n^2/4 \rfloor$

Table 1: Bounds on the second-smallest eigenvalue λ_2 of the Laplacian L, the isoperimetric ratio obtained by the spectral partitioning algorithm with a sweep cut, and the bisection size obtained by a spectral algorithm of Spielman and Teng (see Section B.2) for seven classes of graphs in which all the edge weights and vertex masses are 1. Here, n is the number of vertices and d is the maximum vertex degree. For the three middle entries of the seven, the bounds on the isoperimetric ratio of the sweep cut follow from λ_2 by Cheeger's inequality.

B.1 Spectral Cuts for Seven Classes of Graphs

One way to apply Cheeger's inequality is to compute λ_2 for a particular graph you are interested in, and see what it tells you about the graph's isoperimetric number. (You can usually obtain a yet tighter upper bound by computing the sweep cut as well.) Another way is to observe that there are classes of graphs for which λ_2 is known, or at least bounded. Table 1 lists the values of λ_2 for seven families of graphs. It also tabulates the isoperimetric ratio of the spectral sweep cut and the size of the bisection found by a spectral algorithm of Spielman and Teng described in Section B.2. The results described in this section apply solely to graphs whose edge weights and vertex masses are all 1.

Two particularly important graphs in the table are planar graphs and high-quality tetrahedral meshes. Spectral partitioners are frequently used to decompose finite element meshes to solve partial differential equations on parallel computers [58, 62]. Spielman and Teng [64, 65] prove that every planar graph has $\lambda_2 \leq 8d/n$, where d is the maximum vertex degree and n is the number of vertices in the graph. Cheeger's inequality thus guarantees that there is a cut with isoperimetric ratio at most $4d/\sqrt{n}$. Section B.3 demonstrates that the sweep cut is such a cut.[8] The maximum vertex degree in a planar finite element mesh is usually a small constant, in which case the isoperimetric ratio is in $O(1/\sqrt{n})$.

Spielman and Teng generalize their result to higher dimensions. Say that a tetrahedral mesh has "high quality" if there is a lower bound on some reasonable measure of the quality of each tetrahedron; e.g. suppose every dihedral angle is greater than 5°. (See Spielman and Teng for a formal characterization of mesh quality in terms of *overlap graphs*.) Then the graph representing the edges of the mesh has $\lambda_2 \in O(d/n^{2/3})$. Typically d is a small constant and the isoperimetric ratio of the sweep cut is in $O(1/n^{1/3})$.

Table 1 includes *expander graphs*, which by definition are families of sparse graphs (usually having a constant maximum vertex degree) whose isoperimetric numbers remain bounded above some constant as n increases. By design, they do not have sparse cuts. Expanders are used in robust network design and error-correcting codes. The left inequality (29) provides a way to test whether a graph is an expander: explicitly calculate λ_2 and check that it exceeds some constant.

[8] This is not the strongest known upper bound on the isoperimetric number of a planar graph. Diks, Djidjev, Sykora, and Vrto [13] show that every planar graph has a cut of $2\sqrt{2dn}$ edges such that each subgraph has at least one-third of the vertices, and they give a linear-time algorithm for finding such a cut. This cut implies that the isoperimetric number cannot exceed $6\sqrt{2d/n}$, which is a stronger bound when $d \geq 5$. However, the spectral partitioning algorithm is not guaranteed to find a cut this good. Diks et al. also prove that there is a bisection that cuts at most $(6\sqrt{2} + 4\sqrt{3})\sqrt{dn}$ edges, which beats the bound given here when $d \geq 4$.

B.2 A Better Bisection Method

What if you want to bisect a graph? The roach graph in Section 2.8 demonstrates that spectral partitioning with the median cut can be truly terrible. A sweep cut that minimizes the isoperimetric ratio is more reliable, as it satisfies Cheeger's inequality, but the subgraphs are often unbalanced.

Spielman and Teng [64, 65] analyze an alternative to the median cut: they find a bisection by applying the sweep cut repeatedly. Their algorithm begins with the graph G and two empty vertex sets $\mathcal{V}_1$ and $\mathcal{V}_2$. Repeat the following steps: find a cut for G with small isoperimetric ratio, replace G with the larger of the two subgraphs, and add the vertices of the smaller subgraph to $\mathcal{V}_1$ or $\mathcal{V}_2$, whichever is smaller. Repeat these steps until either $\mathcal{V}_1$ or $\mathcal{V}_2$ has $\lfloor n/2 \rfloor$ vertices; these vertices induce a bisection of the original graph. Spielman and Teng show that if you can find a cut of isoperimetric ratio $h(i)$ or less for every i-vertex subgraph of G, then the weight of the bisection produced by this procedure is at most

$$\int_1^n h(i)\, di.$$

For the middle three classes of graphs in Table 1, this formula produces the upper bounds on the number of bisection edges. The procedure finds bisections of $O(\sqrt{n})$ edges for bounded-degree planar graphs and $O(n^{2/3})$ edges for bounded-degree high-quality tetrahedral meshes. These bounds are asymptotically tight: there are graphs in each class that have no smaller bisection, and such graphs are common in practice. Compare the $O(\sqrt{n})$ bound with the linear-size median cut for the roach graph in Section 2.8.

The disadvantage of this algorithm is that it requires multiple Fiedler vector computations (on matrices of successively smaller sizes), and thus greater computation time.

B.3 Proof of Cheeger's Inequality

Let v_2 be an eigenvector such that $Lv_2 = \lambda_2 M v_2$. Assume that the vertices whose components in v_2 are positive account for at most half the mass. (Otherwise, replace v_2 with $-v_2$.) Let x be the Fiedler vector v_2 with its negative components changed to zero; that is,

$$x_i = \max\{(v_2)_i, 0\}.$$

Observe that $x^\mathsf{T} M x = x^\mathsf{T} M v_2$ (because M is diagonal). The vector $x - v_2$ is a sort of mirror of x: it lists only the negative components of v_2, albeit with their signs changed to positive. The inequality $x^\mathsf{T} L(x - v_2) \leq 0$ holds because every term of $x^\mathsf{T} L(x - v_2)$ is a product of a negative, off-diagonal component of L, a positive component of x, and a positive component of $x - v_2$. Hence,

$$x^\mathsf{T} L x = x^\mathsf{T} L(x - v_2) + x^\mathsf{T} L v_2 \leq x^\mathsf{T} L v_2 = \lambda_2 x^\mathsf{T} M v_2 = \lambda_2 x^\mathsf{T} M x. \tag{30}$$

To find a relationship between λ_2 and the sweep cut, assume without loss of generality that the components of x occur in order from greatest to least. This proof will show that there is a good cut in the positive part of the Fiedler vector, whose vertices have at most half the mass. Let $G_{i\ldots j}$ denote the subgraph of G having vertices i through j, inclusive. The positive-threshold sweep cut has isoperimetric ratio

$$h = \min_{k \geq 1, x_k > 0} \frac{\text{Cut}(G_{1\ldots k}, G_{k+1\ldots n})}{\min\{\text{Mass}(G_{1\ldots k}), \text{Mass}(G_{k+1\ldots n})\}} = \min_{k \geq 1, x_k > 0} \frac{\text{Cut}(G_{1\ldots k}, G_{k+1\ldots n})}{\text{Mass}(G_{1\ldots k})}. \tag{31}$$

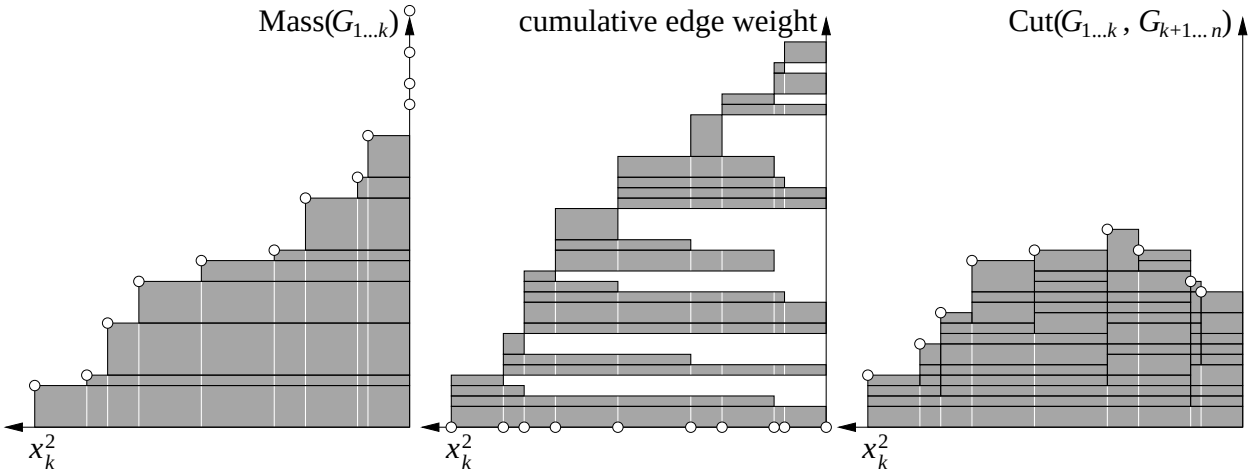

Figure 15: Plots of $\mathrm{Mass}(G_{1...k})$ and $\mathrm{Cut}(G_{1...k}, G_{k+1...n})$ as functions of each vertex's squared Fiedler component x_k^2. The vertices appear as circles, ordered from left to right by increasing index k and decreasing component x_k. Vertices with negative components are plotted at $x_k = 0$, and most of them are not shown. In the center plot, each horizontal bar represents an edge of G. The right plot is obtained by vertically compacting the center plot.

Figure 15 plots $\mathrm{Mass}(G_{1...k})$ and $\mathrm{Cut}(G_{1...k}, G_{k+1...n})$ as functions of x_k^2. It follows from Equation (31) that the shaded area under the latter plot is at least h times the shaded area under the former plot.

In the plot of $\mathrm{Mass}(G_{1...k}) = \sum_{i=1}^{k} M_{ii}$, each horizontal (black-bordered) rectangle has a height equal to the corresponding vertex's mass M_{ii}. The shaded area under the plot can be accounted for by summing either the vertical (white-bordered) rectangles or the horizontal rectangles, giving the identity

$$\begin{aligned} \sum_{k=1}^{n} \mathrm{Mass}(G_{1...k}) \left(x_k^2 - x_{k+1}^2\right) &= \sum_{i=1}^{n} M_{ii} x_i^2 \\ &= x^{\mathsf{T}} M x. \end{aligned} \tag{32}$$

In the center plot of Figure 15, each horizontal (black-bordered) rectangle represents an edge whose weight is the height of the rectangle. The weight of a threshold cut is the sum of the heights of the rectangles that cross the threshold, so vertical compaction of the shaded area yields the plot of $\mathrm{Cut}(G_{1...k}, G_{k+1...n})$. The shaded area under that plot can be accounted for by summing either its vertical rectangles or the horizontal rectangles in the center plot, giving the identity

$$\sum_{k \geq 1, x_k > 0} \mathrm{Cut}(G_{1...k}, G_{k+1...n}) \left(x_k^2 - x_{k+1}^2\right) = \sum_{(i,j) \in \mathcal{E}} w_{ij} \left|x_i^2 - x_j^2\right|. \tag{33}$$

Substituting Equations (31) and (32) into Equation (33) gives

$$\begin{aligned} \sum_{(i,j) \in \mathcal{E}} w_{ij} \left|x_i^2 - x_j^2\right| &\geq h \sum_{k \geq 1, x_k > 0} \mathrm{Mass}(G_{1...k}) \left(x_k^2 - x_{k+1}^2\right) \\ &= h x^{\mathsf{T}} M x. \end{aligned} \tag{34}$$

Observe that the left-hand side of Inequality (34) bears a resemblence to Expression (2). The latter helps to impose an upper bound on the former, but the relationship is a bit opaque. By the Cauchy–Schwarz

inequality and then Equation (3),

$$\begin{aligned}
\sum_{(i,j)\in\mathcal{E}} w_{ij}\left|x_i^2 - x_j^2\right| &= \sum_{(i,j)\in\mathcal{E}} w_{ij}|x_i - x_j|(x_i + x_j) \\
&\leq \left(\sum_{(i,j)\in\mathcal{E}} w_{ij}(x_i - x_j)^2\right)^{1/2}\left(\sum_{(i,j)\in\mathcal{E}} w_{ij}(x_i + x_j)^2\right)^{1/2} \\
&\leq \sqrt{x^\mathsf{T} L x}\left(2\sum_{(i,j)\in\mathcal{E}} w_{ij}\left(x_i^2 + x_j^2\right)\right)^{1/2} \\
&= \sqrt{x^\mathsf{T} L x}\left(2\sum_{i=1}^{n}\sum_{j:(i,j)\in\mathcal{E}} w_{ij}x_i^2\right)^{1/2} \\
&= \sqrt{x^\mathsf{T} L x}\left(2\sum_{i=1}^{n} L_{ii}x_i^2\right)^{1/2} \\
&= \sqrt{2x^\mathsf{T} L x\, x^\mathsf{T}\overline{L}x},
\end{aligned}$$

where $\overline{L}$ is the diagonal matrix that has the same diagonal components as L.

Substituting this inequality into Inequality (34), then substituting Inequality (30) into that, gives

$$\begin{aligned}
h &\leq \sqrt{2\frac{x^\mathsf{T} L x}{x^\mathsf{T} M x}\,\frac{x^\mathsf{T}\overline{L}x}{x^\mathsf{T} M x}} \\
&\leq \sqrt{2\lambda_2\frac{x^\mathsf{T}\overline{L}x}{x^\mathsf{T} M x}} \\
&\leq \sqrt{2\lambda_2 \max_{1\leq i\leq n}\frac{L_{ii}}{M_{ii}}},
\end{aligned}$$

and therefore

$$\min_{1\leq k<n}\frac{\mathrm{Cut}(G_{1\ldots k}, G_{k+1\ldots n})}{\min\{\mathrm{Mass}(G_{1\ldots k}), \mathrm{Mass}(G_{k+1\ldots n})\}} \leq \sqrt{2\lambda_2 \max_{1\leq i\leq n}\frac{L_{ii}}{M_{ii}}}.$$

B.4 Higher-Order Cheeger Inequalities

Kwok, Lau, Lee, Oveis Gharan, and Trevisan [42] show that the isoperimetric number of a graph is in $O(k\lambda_2/\sqrt{\lambda_k})$ for any $k \geq 2$. Moreover, a sweep cut achieves this bound, despite the fact that the spectral partitioning algorithm does not use any information about the eigenvalues and eigenvectors besides λ_2 and v_2. The bound is an improvement on Cheeger's inequality when there is a sufficiently large gap between λ_2 and a subsequent eigenvalue. It implies that spectral partitioning is not only an $O(1/\sqrt{\lambda_2})$-approximation algorithm for the sparsest cut, but also an $O(k/\sqrt{\lambda_k})$-approximation algorithm. In particular, if a family of graphs has the property that for some fixed $k \geq 2$, λ_k remains constant as the graph size grows, then spectral partitioning is an $O(1)$-approximation algorithm for that family.

See Section E.6 for a related bound governing partitioning a graph into multiple subgraphs.

C Vertex Separators

A *vertex separator* for a graph G is a set of vertices whose removal from G disconnects it into two or more connected components. (Removing a vertex also entails removing the edges adjoining the vertex.) Many divide-and-conquer algorithms on graphs, including nested dissection [47], find a vertex separator and recurse on the subgraphs. Some parallel computing applications also use vertex separators, which represent data that must be shared by two or more processors. (In the former application, neither subgraph includes the separator vertices; in the latter application, both subgraphs do.) Usually, the goal is to find the smallest vertex separator whose removal separates the graph into two subgraphs of roughly equal mass.

The literature of spectral graph theory includes inequalities governing the sizes of vertex separators. Throughout this section, suppose that every edge weight and vertex mass is 1 and G is not a complete graph. Recall from Section 1.5 that Fiedler [22] shows that the smallest vertex separator (with no balance constraint) has at least λ_2 vertices, where λ_2 is the second-least eigenvalue of G's Laplacian matrix. Pothen, Simon, and Liou [58] prove an inequality akin to Inequality (21) in Section 2.7. They show that if G_1 is a subgraph of G covering ψn of the n vertices for some $\psi \in (0, 1)$, then a vertex separator whose removal disconnects G_1 from the remainder of G comprises at least $4\psi(1-\psi)n/\left((d/\lambda_2) + 4\psi - 1\right)$ vertices, where d is the maximum degree among G's vertices.

Every planar graph has a small vertex separator, even if it has no small edge separator. An instructive extreme case is the *star graph* in which one central vertex adjoins the other $n-1$ vertices, which do not adjoin each other. A star graph has a minimum edge bisection of $\lfloor n/2 \rfloor$ edges, but a vertex bisection of one vertex. Djidjev [15] shows that every planar graph can be disconnected into two or more subgraphs, each having no more than two thirds of the vertices, by the removal of $\sqrt{6n}$ vertices. (A famous earlier paper by Lipton and Tarjan [48] proved the first $O(\sqrt{n})$ bound, namely, $\sqrt{8n}$ vertices.)

Pothen, Simon, and Liou [58] suggest an algorithm for computing a vertex separator by first finding an edge separator, then converting it into a vertex separator by finding a minimum vertex cover. (Again we assume that all edges have weight 1; this approach does not seem to generalize to a model where some vertices cost more to remove than others.) Let $\mathcal{S} \subseteq \mathcal{E}$ be an edge separator that cuts $G = (\mathcal{V}, \mathcal{E})$ into subgraphs G_1 and G_2, as illustrated in Figure 16(a), and let $\mathcal{W} \subseteq \mathcal{V}$ be the set of endpoints of edges in $\mathcal{S}$. The graph $(\mathcal{W}, \mathcal{S})$ is bipartite, because every edge in $\mathcal{S}$ connects a vertex of G_1 to a vertex of G_2. Because $\mathcal{S}$ is an edge separator, removing one endpoint of each edge in $\mathcal{S}$ from G disconnects G (unless *all* the vertices in G_1 or G_2 are removed). A vertex in $\mathcal{W}$ can be incident on multiple edges in $\mathcal{S}$. The smallest subset of $\mathcal{W}$ that adjoins every edge in $\mathcal{S}$ is called the *minimum bipartite vertex cover* of $\mathcal{S}$, illustrated in Figure 16(e).

To compute a minimum bipartite vertex cover, first compute a *maximum bipartite matching* of $\mathcal{S}$—the largest subset of $\mathcal{S}$ such that no two edges share an endpoint, illustrated in Figure 16(b). Clearly, every vertex cover must include at least one endpoint of each edge in the matching. By Kőnig's Theorem [39], each minimum vertex cover contains *only* one endpoint of each edge in the matching.

A maximum bipartite matching can be found in $O(|\mathcal{S}|\sqrt{|\mathcal{W}|})$ time by a network flow algorithm of Hopcroft and Karp [36], but some asymptotically slower algorithms are faster in practice. Cherkassky, Goldberg, Martin, Setubal, and Stolfi [10] performed an experimental study of bipartite matching implementations, and recommend a variant of the push-relabel algorithm of Goldberg and Tarjan [26]. Pothen et al. [58] report that in practice, several different bipartite matching algorithms appear to run in $O(|\mathcal{S}| + |\mathcal{W}|)$ time on the edge separators typical of scientific computing applications.

To produce a minimum bipartite vertex cover from a maximum matching in $O(|\mathcal{S}| + |\mathcal{W}|)$ time, initialize the vertex cover to an empty set of vertices and iterate the following steps. First, for each vertex $w \in \mathcal{W}$ that does not participate in the matching, add to the cover the *other* endpoint of each edge of $\mathcal{S}$ incident on

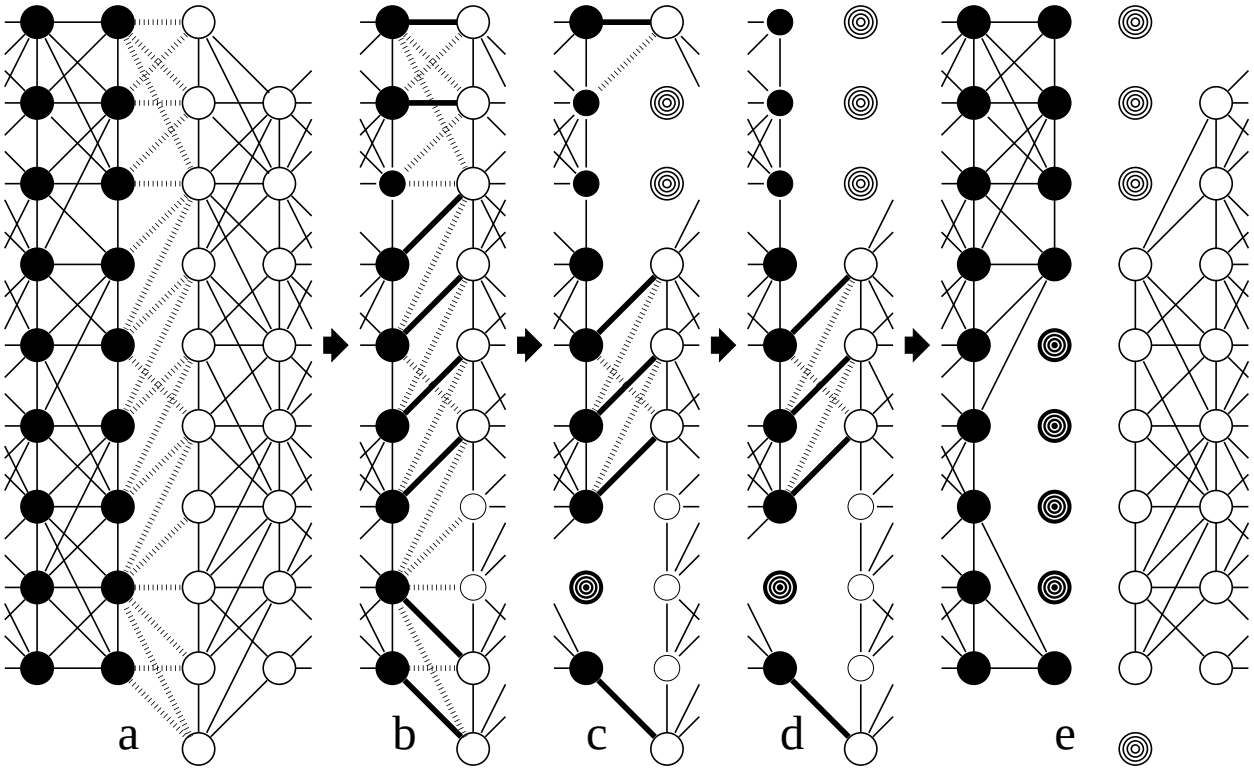

Figure 16: Computation of a vertex separator (e) from an edge separator (a). (b) A maximum bipartite matching in the edge separator. (c, d) A greedy algorithm identifies vertices (drawn as targets) that are in every minimum vertex cover. (e) The remaining vertices in the cover are chosen to help balance the subgraphs.

w. (Because w will not be in the cover, all its neighbors must be in the cover.) Second, remove w and its neighbors from $\mathcal{W}$, and remove every edge adjoining them from both $\mathcal{S}$ and the matching, as illustrated in Figure 16(c). Removing these edges from the matching may leave behind some newly unmatched vertices. Repeat these two steps until no further changes occur, as illustrated in Figure 16(d).

There may still be edges left in the matching. Partition the graph induced by the surviving edges of the matching into its connected components. For each of these components, we can choose between adding to the cover the endpoints in G_1 or the endpoints in G_2. Pothen et al. use this flexibility to help balance the subgraphs, as illustrated in Figure 16(e). Because a vertex separator removes vertices from G, a balanced edge separator might yield an unbalanced vertex separator. The freedom to choose some of the vertices in the separator often suffices to restore the balance, and sometimes helps to produce subgraphs that are better balanced than G_1 and G_2. However, there is no guarantee that the vertex separator is optimal, even if the edge separator is.

The number of vertices in a vertex separator found this way never exceeds the number of edges in the edge separator provided as input. Therefore, Cheeger's inequality guarantees the same upper bound on the size of the vertex separator as it guarantees for the edge separator. The lower bound does not hold, though, as the vertex separator might be much smaller than the edge separator. Therefore, although the spectral partitioning algorithm often finds a good vertex separator, it cannot be considered an approximation algorithm for finding the smallest vertex separator.

D Better Partitions from Multiple Vectors

The idea to use multiple eigenvectors, and not just v_2, for clustering and partitioning dates right back to Hall's 1970 paper [31], and has been revisited frequently by researchers. Just as the Fiedler vector positions each vertex on the number line, we can use k eigenvectors to assign each vertex *spectral coordinates* in a k-dimensional space, as illustrated in Figure 17. Vertices that are strongly linked by the graph tend to be geometrically close to each other in spectral space. We can partition the points in spectral space with a geometric clustering algorithm [17, 49] or a geometric partitioning algorithm [51]. Computing several eigenvectors isn't much more expensive than computing one; a better partition can cost only a little extra.

D.1 Exploiting Multiple Eigenvectors through Vector Partitioning

Alpert, Kahng, and Yao [2] offer a mathematically compelling approach that uses eigenvectors to reduce the graph partitioning problem to another discrete optimization problem called *vector partitioning*. The vector partitioning problem is also NP-hard, but heuristics work well for it.

For simplicity, use a 0-1 indicator vector x, so $x_i = 1$ if vertex i is in subgraph G_1, and $x_i = 0$ otherwise. (It is irrelevant that the eigenvectors aren't a good fit to 0-1 vectors.) Let $V = [v_1, v_2, \ldots, v_n]$ be the $n \times n$ matrix whose columns are the eigenvectors of the eigensystem $Lv = \lambda Mv$, each scaled so that $v_i^\mathsf{T} M v_i = 1$. Let Λ be the diagonal $n \times n$ matrix whose diagonal entries are the corresponding eigenvalues; thus

$$LV = MV\Lambda \quad \text{and} \quad V^\mathsf{T} M V = I.$$

Write the indicator vector as a linear combination of eigenvectors,

$$x = Va.$$

The vector $a \in \mathbb{R}^n$ of coefficients for that linear combination can be computed from x by the identity

$$a = V^\mathsf{T} M V a = V^\mathsf{T} M x.$$

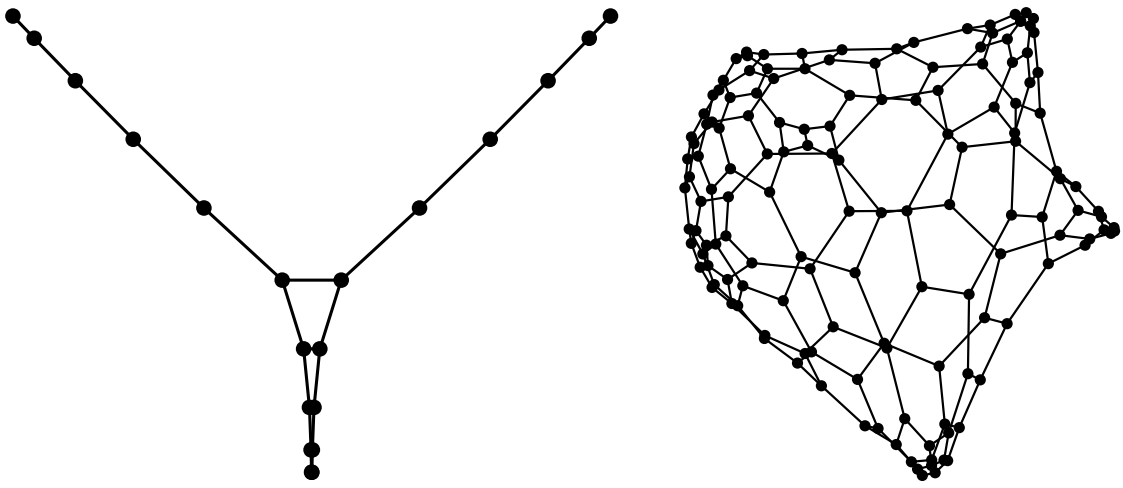

Figure 17: Two graphs with their vertices positioned by the eigenvectors v_2 (horizontally) and v_3 (vertically). At left is the roach graph from Section 2.8. At right is a graph representing the surface of a ball.

We wish to minimize $\text{Cut}(G_1, G_2)$, which by Equation (3) is

$$\begin{aligned} x^\mathsf{T} L x &= a^\mathsf{T} V^\mathsf{T} L V a \\ &= a^\mathsf{T} V^\mathsf{T} M V \Lambda a \\ &= a^\mathsf{T} \Lambda a \\ &= \left|\Lambda^{1/2} a\right|^2 \\ &= \left|\Lambda^{1/2} V^\mathsf{T} M x\right|^2 \\ &= \left|\sum_{j \in G_1} s_j\right|^2, \end{aligned} \tag{35}$$

where s_j is the jth column of $\Lambda^{1/2} V^\mathsf{T} M$, namely,

$$s_j = m_j \left[\sqrt{\lambda_1} V_{j1},\ \sqrt{\lambda_2} V_{j2}, \ldots,\ \sqrt{\lambda_n} V_{jn}\right]^\mathsf{T},$$

which serves as an n-coordinate *spectral vector* for vertex j. Observe that each eigenvector is scaled by the square root of its associated eigenvalue, and each vertex is assigned one coordinate from each scaled eigenvector, scaled by the vertex's mass.

Vector partitioning is the problem of choosing a subset of G's vertices that sum to a specified mass and minimize the objective function (35)—the length of the sum of their spectral vectors. Because graph partitioning reduces to vector partitioning, the latter is also NP-hard.

At first sight, the vector partitioning problem appears as menacing as the graph partitioning problem. It is difficult to see a heuristic for choosing a subset of vectors whose sum is as close to zero as possible; the problem is reminiscent of the famous NP-complete subset-sum problem. However, if the goal were to *maximize* the objective function, the problem would admit simple heuristic solutions: we would try to choose long vectors that point in roughly the same direction. Fortunately, the problem can be recast in this form, by changing how the eigenvectors are scaled.

Let μ be the mass you want G_1 to have, and impose the constraint

$$\mu = \text{Mass}(G_1) = x^\mathsf{T} M x = a^\mathsf{T} V^\mathsf{T} M V a = a^\mathsf{T} a.$$

For an arbitrary constant ξ, minimizing the cut weight is equivalent to maximizing

$$\begin{aligned} \xi\mu - \text{Cut}(G_1, G_2) &= \xi x^\mathsf{T} M x - x^\mathsf{T} L x \\ &= \xi a^\mathsf{T} a - a^\mathsf{T} \Lambda a \\ &= a^\mathsf{T} (\xi I - \Lambda) a \\ &= \left|(\xi I - \Lambda)^{1/2} V^\mathsf{T} M x\right|^2 \\ &= \left|\sum_{j \in G_1} \hat{s}_j\right|^2, \end{aligned} \tag{36}$$

where the spectral vector $\hat{s}_j$ is the jth column of $(\xi I - \Lambda)^{1/2} V^\mathsf{T} M$, namely

$$\hat{s}_j = m_j \left[\sqrt{\xi - \lambda_1} V_{j1},\ \sqrt{\xi - \lambda_2} V_{j2}, \ldots,\ \sqrt{\xi - \lambda_n} V_{jn}\right]^\mathsf{T}.$$

Alpert et al. experimented with several values of ξ and recommend $\xi = \lambda_n + \lambda_2$, for which all the coefficients are positive. Note that $\sqrt{\xi - \lambda_1} V_{j1} = \sqrt{\xi}$ (because $v_1 = \mathbf{1}$ and $\lambda_1 = 0$), which can help speed up the computation of (36). If all the vertex masses are equal, every spectral vector has the same first coordinate, and the first coordinate of each vector may be dropped.

Our modified vector partitioning problem is to choose a subset of G's vertices that sum to a specified mass and *maximize* the length of the sum of the spectral vectors (36). An exact solution of this NP-hard vector partitioning problem gives an exact solution of the graph partitioning problem. However, it is usually impractical to compute all n eigenvectors (especially for large, sparse matrices), and the high-frequency eigenvectors are of diminishing value for partitioning. Alpert et al. compute only the first k eigenvectors for some small k, so each vertex has a k-dimensional spectral vector. Truncate V to be an $n \times k$ matrix, holding only the first k eigenvectors; truncate Λ to be $k \times k$; and truncate each spectral vector $\hat{s}_j$ to k coordinates. Observe that the identities $LV = MV\Lambda$ and $V^\mathsf{T} MV = I$ still hold for these truncated matrices.

Unfortunately, it is unlikely that any 0-1 indicator vectors (except $\mathbf{0}$ and $\mathbf{1}$) are linear combinations of the first k eigenvectors, and Equations (35) and (36) do not hold for vectors that are not 0-1 indicator vectors. But every indicator vector can be projected into the subspace spanned by those eigenvectors. For any 0-1 indicator vector x, the vector $a = V^\mathsf{T} Mx$ specifies the coefficients of the projection of x onto the subspace spanned by the columns of V, and $\bar{x} = Va$ is the projection itself. (If V included all n eigenvectors, $\bar{x}$ would equal x.) The optimal vector partition of the truncated spectral vectors maximizes

$$\begin{aligned} \left| \sum_{j \in G_1} \hat{s}_j \right|^2 &= \xi a^\mathsf{T} a - a^\mathsf{T} \Lambda a \\ &= \xi a^\mathsf{T} V^\mathsf{T} MVa - a^\mathsf{T} V^\mathsf{T} MV\Lambda a \\ &= \bar{x}^\mathsf{T} (\xi M - L)\bar{x}. \end{aligned}$$

If $\bar{x}$ approximates x well, then the algorithm approximately maximizes $\xi\mu - \text{Cut}(G_1, G_2)$. The accuracy of this approximation depends on how close x is to the subspace the eigenvectors span.

Alpert et al. use a greedy incremental algorithm to guess a solution to the truncated vector partitioning problem. Suppose every vertex has mass 1. Their algorithm begins by placing the longest vector in G_1. Then it repeatedly adds the vector that maximizes the sum of vectors so far until μ vectors are chosen. It is easy to adapt this heuristic to the case where vertices have different masses: each iteration greedily chooses the vector that maximizes the vector sum divided by the total mass so far until mass μ has accumulated. Unfortunately, this heuristic takes $O(n^2)$ time.

A faster heuristic follows from the following observation: if you knew the direction of the maximum vector sum, you could determine the vectors in the sum in expected $O(n)$ time by using the quickselect algorithm to select the spectral vectors whose components in that direction are greatest. If you guess the direction and are wrong by an angle of θ, you will choose vectors that approximate the length of the maximum vector sum to within a factor of $\cos\theta$. If the dimension k of the spectral vectors is small, there is a polynomial-time approximation scheme for vector partitioning: discretize the set of directions by choosing a finite subset such that every direction is within some angle θ of a direction in the subset, and try every direction in the subset. The number of directions you must try grows exponentially with k but is independent of the number n of spectral vectors, so the running time is linear in n. (Note that this heuristic does not give an approximation scheme for graph partitioning, because it does not use all the eigenvectors and because it approximates $\xi\mu - \text{Cut}(G_1, G_2)$, not $\text{Cut}(G_1, G_2)$).

In summary, the algorithm computes the first k eigenvalues and eigenvectors of $Lv = \lambda Mv$, computes one k-coordinate spectral vector for each vertex, and heuristically chooses spectral vectors of fixed total

mass μ whose sum has approximately maximum length. If we use just one eigenvector v_2, these heuristics simply duplicate the standard spectral partitioning algorithm. Alpert et al. give evidence that the quality of the partition tends to improve with the number of eigenvectors, at least up to ten eigenvectors (the largest number they tried).

Why is the NP-hard problem of vector partitioning more tractable than the NP-hard problem of graph partitioning? Alpert et al. [2] explain that vector partitioning

> may not seem much different from a graph traversal, e.g., choose some vertex to be in its own cluster and iteratively add vertices to the cluster to minimize the cluster degree. However, with our approach each vector contains global partitioning information; the set of edges connected to any vertex in the original graph representation comprises strictly local information.

Vector partitioning accommodates different objective functions—for instance, the minimum cut, the sparsity, or the isoperimetric ratio. We can order the vertices with a greedy heuristic, then apply the sweep cut described in Section 2.2 as usual. This has the effect of trying many different values of μ, approximately minimizing the cut weight for each one, and choosing from these options the cut that minimizes the sparsity or isoperimetric ratio.

D.2 Exploiting Multiple Isoperimetric Solutions through Vector Partitioning

The method of Alpert et al. doesn't need eigenvectors. It can use any orthogonal set of vectors that form a good basis for constructing partitions. One way to construct such a basis, faster than computing eigenvectors, is to compute several isoperimetric partitions, grounding a different vertex at zero potential for each partition.

Let B (for "basis") be a nonsingular $n \times n$ matrix whose column vectors are mutually M-orthogonal and are scaled so that $B^\mathsf{T} M B = I$. (B is a basis for $\mathbb{R}^n$; we will consider subspaces with much smaller bases shortly.) To be useful, the basis vectors are not 0-1 vectors; they might be eigenvectors or isoperimetric solution vectors prior to rounding. Let x be a 0-1 indicator vector. Write it as a linear combination of basis vectors, $x = Ba$. Observe that $a = B^\mathsf{T} M B a = B^\mathsf{T} M x$, so the coefficients of a are easily computed from x.

Let us revisit the vector partition maximization problem of Section D.1. Let μ be the mass you want G_1 to have, and constrain $x^\mathsf{T} M x = \mu$. Let ξ be an arbitrary constant. Minimizing the cut weight is equivalent to maximizing

$$\begin{aligned}
\xi\mu - \mathrm{Cut}(G_1, G_2) &= \xi x^\mathsf{T} M x - x^\mathsf{T} L x \\
&= x^\mathsf{T}(\xi M - L)x \\
&= a^\mathsf{T} B^\mathsf{T}(\xi M - L)Ba \\
&\equiv a^\mathsf{T} J a \\
&= |J^{1/2} a|^2 \\
&= |J^{1/2} B^\mathsf{T} M x|^2 \\
&= \left| \sum_{j \in G_1} t_j \right|^2 , \qquad (37)
\end{aligned}$$

where $J = B^\mathsf{T}(\xi M - L)B \in \mathbb{R}^{n \times n}$; $J^{1/2} \in \mathbb{R}^{n \times n}$ is the upper triangular square root of J, computed by Cholesky decomposition [11, 27]; and $t_j \in \mathbb{R}^n$ is the jth column of $J^{1/2} B^\mathsf{T} M$.

Call t_j the *fingerprint vector* for vertex j. Equation (37) shows how to reduce the graph partitioning problem to a different vector partitioning problem for every M-orthogonal basis B you can think of. For example, if the columns of B are the eigenvectors of L, we recover the Alpert–Kahng–Yao method of Section D.1: $B = V$, $J = \xi I - \Lambda$, and t_j is a spectral vector. In general, though, J is dense.

Of course, it is too expensive to use a basis with full rank; we want to use a small number k of well-chosen basis vectors. Then B is an $n \times k$ matrix and J is a $k \times k$ matrix, so it is inexpensive to explicitly compute J and $J^{1/2}$. The effect is to truncate the fingerprint vectors to their first k coordinates.

Unfortunately, few 0-1 indicator vectors lie in the subspace spanned by the k basis vectors. As discussed in Section D.1, the objective function (37) only approximates the function we actually want to optimize. Specifically, the algorithm maximizes $\bar{x}^{\mathsf{T}}(\xi M - L)\bar{x}$, where $\bar{x} = BB^{\mathsf{T}}Mx$ is the projection of a 0-1 indicator vector x onto the subspace spanned by the basis.

The algorithm, then, is to choose a basis B and a constant ξ, compute the $k \times k$ matrix J, its Cholesky decomposition $J^{1/2}$, and the fingerprint vectors t_j, then apply the heuristics described in Section D.1 to find an approximately maximum partition of the fingerprint vectors. So that $J^{1/2}$ exists, ξ should be chosen greater than the greatest eigenvalue of the generalized eigensystem $Lv = \lambda Mv$.

How should we choose a basis B? We want the subspace spanned by the basis to contain vectors that round to good partitions, and to include enough variety that it is likely to come near the optimal partition. Therefore, we should choose basis vectors that round to good partitions, but are quite different from each other. For instance, we may take the (unrounded) vectors produced by several runs of isoperimetric partitioning, each time grounding a different vertex at zero potential. Probably these vectors are not M-orthogonal to each other; use Gram–Schmidt conjugation [27, Section 5.2.8] to convert them into an M-orthogonal basis for the same subspace, then scale the basis vectors so that $B^{\mathsf{T}}MB = I$.

Which vertex should be grounded at zero to generate each new indicator vector? Because the ground vertex is always assigned the least voltage, it seems reasonable to ground a vertex that has not come close to being an extreme (least or greatest) vertex in any of the previous indicator vectors. For example, the second vector might be generated by grounding the median vertex of the first vector.

E Beyond Bisection: k-Subgraph Partitions

The most common way of partitioning a graph into more than two subgraphs is to cut the graph and its subgraphs repeatedly—an approach called *recursive partitioning* or, if all the subgraphs are to have equal mass, *recursive bisection*—but it is not the most reliable approach, because the first cut is not chosen with later cuts in mind.

This section considers five other approaches and illustrates the variety among them. We distinguish *partitioning with prescribed subgraph masses* from *graph clustering*, wherein the user permits the partitioner to choose the subgraph masses that yield the most natural decomposition. The first and most flexible approach, vector partitioning (introduced in Section D), uses an arbitrary number of eigenvectors or isoperimetric vectors to cut a graph into an arbitrary number of subgraphs. It works with all the standard cut criteria, and works equally well for clustering and for prescribed-mass subgraphs. Two algorithms designed mainly for clustering, by Ng, Jordan, and Weiss [54] and Yu and Shi [70] (Sections E.2 and E.5, respectively), use the first k eigenvectors to cut a graph into k subgraphs. The former method can be viewed as a vector partition based on the angles between the spectral vectors, whereas the latter method can be viewed as a vector partition based on rotating the spectral vectors to turn them into indicator vectors that minimize the sparsity. By contrast, an algorithm by Hendrickson and Leland [33] (Section E.4) is specialized for balanced multisection (subgraphs of equal size). It produces 2^k subgraphs from k eigenvectors.

Most methods of partitioning a graph into three or more subgraphs use multiple indicator vectors. Define an *indicator matrix* $X \in \mathbb{R}^{n\times k}$ such that $X_{ji} = c_{\text{yes}}$ if vertex j is in subgraph G_i, and $X_{ji} = 0$ otherwise. The columns of X are indicator vectors for the k subgraphs, written $X = [X_{*1}, X_{*2}, \ldots, X_{*k}]$. Each row of X should have one entry of c_{yes}, and all other entries zero. Rewrite Equation (3) as

$$\text{Cut}(G_i, G\backslash G_i) = \frac{X_{*i}^{\mathsf{T}} L X_{*i}}{c_{\text{yes}}^2}.$$

The sum of this quantity over all the subgraphs counts each cut edge twice, so the weight of a cut that decomposes G into k subgraphs is

$$\text{Cut}(G_1, G_2, \ldots, G_k) = \sum_{i=1}^{k} \frac{X_{*i}^{\mathsf{T}} L X_{*i}}{2c_{\text{yes}}^2} = \frac{\text{trace}\left(X^{\mathsf{T}} L X\right)}{2c_{\text{yes}}^2}. \tag{38}$$

The trace of a square matrix is the sum of its diagonal components. We will occasinally need the identity

$$\text{trace}(AB) = \text{trace}(BA), \tag{39}$$

which holds for all square AB, even if A and B are not square (implying that AB and BA have different sizes). The proof is simple: $\text{trace}(AB) = \sum_i (AB)_{ii} = \sum_i \sum_j A_{ij} B_{ji} = \sum_j \sum_i B_{ji} A_{ij} = \text{trace}(BA)$.

E.1 Vector Partitioning

Alpert, Kahng, and Yao [2] express the k-subgraph partitioning problem as a k-way vector partitioning problem. For best results, use k or more eigenvectors. Let X be a 0-1 indicator matrix, so $c_{\text{yes}} = 1$. Following Equations (36) and (37), for an arbitrary constant ξ, minimizing the cut weight (38) is equivalent to maximizing

$$\xi\,\text{Mass}(G) - \sum_{i=1}^{k} X_{*i}^{\mathsf{T}} L X_{*i} = \sum_{i=1}^{k} \left| \sum_{j\in G_i} t_j \right|^2, \tag{40}$$

where t_j is the spectral vector (from Section D.1) or fingerprint vector (from Section D.2) for vertex j.

The objective function (40) is the sum of squares of the lengths of k vectors, one vector for each subgraph. Each vector is the sum of the fingerprint vectors of the vertices in the subgraph. Therefore, the goal is to partition the vertices so that these vectors are as long as possible, and the partition is balanced as requested. The same greedy heuristics mentioned in Section D.1 work for k-way vector partitioning too. See Alpert et al. [2] for elaboration.

E.2 Spectral Clustering by Spectral Vector Directions

Ng, Jordan, and Weiss [54] suggest a spectral clustering method (refining a suggestion of Shi and Malik [61]) that uses multiple eigenvectors to partition a graph into multiple subgraphs, also called *clusters*. It works best if the number of eigenvectors (including the seemingly useless $v_1 = \mathbf{1}$) is at least as great as the intended number of clusters. The method is related to vector partitioning (see Section D), but the clusters are chosen by examining the similarity between spectral vectors, especially their directions, rather than the lengths of sums of spectral vectors. Both groups assign each vertex a mass according to the normalized cut criterion described in Section A, but their ideas generalize to arbitrary positive vertex masses.

Begin by computing the first k eigenvectors $v_1, \ldots, v_k$ of the generalized eigensystem $Lv = \lambda Mv$, where M is the diagonal mass matrix defined in Section 2.3. Scale these eigenvectors so that $v_i^\mathsf{T} M v_i = 1$ and collect them in an $n \times k$ matrix $V = [v_1, \ldots, v_k]$. The generalized eigenvectors are mutually M-orthogonal, so $V^\mathsf{T} MV = I$. Call row i of V the *spectral vector* for vertex i. (Note that these spectral vectors are different from those of Alpert et al. [2], because the eigenvectors are scaled differently.) We will see that it makes sense to cluster vertices together if their spectral vectors are similar.

Following Ng et al., a helpful way to understand the method is to imagine the extreme case where we are given a graph G with exactly k connected components and asked to partition it into k clusters. The obvious answer is to make each connected component be a cluster. Recall from Section 1.3 that for each connected component there is an eigenvector with eigenvalue zero, namely, the vector whose components are 1 for every vertex in the component and 0 for every vertex in the other components. Call these eigenvectors the *connected component eigenvectors*. Let the $n \times k$ *connected component matrix* W be a matrix whose columns are the connected component eigenvectors, scaled so that $w^\mathsf{T} Mw = 1$ for each column w of W. By construction, each row of W has just one nonzero component, so the connected component eigenvectors are mutually M-orthogonal and $W^\mathsf{T} MW = I$.

However, every linear combination of the connected component eigenvectors is also an eigenvector with eigenvalue zero. Hence an algorithm that computes V might compute eigenvectors different from the connected component eigenvectors. In practice, most implementations will set $v_1 \propto \mathbf{1}$ to save eigenvector computation time, so no column of V will be a connected component eigenvector, though every column of V will be a linear combination of them. Express this fact by writing $V = WR$, where $R \in \mathbb{R}^{k \times k}$ collects the coefficients of the linear combinations. (The columns of V span the same k-dimensional subspace as the columns of W, namely, the null space of the Laplacian L.)

Observe that $I = V^\mathsf{T} MV = R^\mathsf{T} W^\mathsf{T} MWR = R^\mathsf{T} R$, so R is an orthonormal matrix. It follows that R's effect is to rotate (or reflect) the rows of W in k-dimensional space to yield the rows of V—the spectral vectors. The computed spectral vectors (rows of V) are simply the connected component spectral vectors (rows of W) after jointly undergoing a rigid rotation in $\mathbb{R}^k$. We desire an algorithm that is insensitive to the choice of rotation, which is arbitrary, and sensitive to the relationships between spectral vectors, which are stable.

If two vertices are in the same connected component of G, their rows in W are identical. Otherwise, their rows of W are mutually orthogonal, because each row of W has just one nonzero component. Therefore, V

has the same properties: two vertices have the same spectral vector if they are in the same connected component, and orthogonal spectral vectors otherwise. (Vertices in the same component would have different spectral vectors if we computed more than k eigenvectors.)

The foregoing discussion holds only if G has k connected components. But imagine modifying G by adding new edges with tiny positive weights that render G connected. Because the new edge weights are tiny, the eigenvectors are perturbed only a little. Typically, no two vertices have the same spectral vector anymore; however, the spectral vectors are clustered around the pre-perturbation vectors.

This thought experiment suggests that we should cluster the spectral vectors according to their similarity. Shi and Malik cluster the vectors with any standard geometric clustering method, such as Lloyd's algorithm for k-means clustering [49]. Ng et al. cluster the vectors according to their directions: vectors that are roughly parallel belong in the same cluster, and vectors that are roughly orthogonal belong in different clusters. They achieve this by normalizing the rows of V to have unit length—in other words, they project each row onto the unit sphere. Then they cluster the points on the sphere with any standard geometric clustering method. They discuss how to initialize the k-means clustering algorithm by exploiting the knowledge that different clusters are expected to have roughly orthogonal spectral vectors.

Whereas Shi and Malik use the eigenvectors of the generalized eigensystem, Ng et al. use the unit eigenvectors of the normalized Laplacian $M^{-1/2}LM^{-1/2}$, which has the effect of premultiplying the generalized eigenvectors by $M^{1/2}$, or multiplying row i of V by the mass M_{ii} of vertex i. Therefore, two vertices that have the same spectral vector in the generalized eigensystem will, if their masses differ, have spectral vectors of different lengths in the eigensystem of the normalized Laplacian. Ng's variant of the clustering algorithm subsequently normalizes the rows of V to have unit length, so the two vertices again come into harmony, but the information encoded in the spectral vector lengths is discarded. This raises the question of whether Shi and Malik's variant benefits from retaining that information, or whether the directions of the spectral vectors are more robust cues for geometric clustering.

Note that the clustering algorithm does not compute W or R. By contrast, Section E.5 presents a method that refines the clusters by attempting to find a suitable choice of R.

E.3 Constraining the Indicator Matrix by Projection

Rendl and Wolkowicz [60] describe a spectral multipartitioning algorithm that uses the eigenvectors $v_2, \ldots, v_k$ to partition a graph into k subgraphs of prescribed masses that approximately minimize the objective function (38). They constrain each vertex to be in exactly one subgraph by projecting the indicator matrix onto a subspace where the constraint always holds. They formulate their algorithm in terms of a graph adjacency matrix; the presentation here follows Pothen [57], who recasts their algorithm in terms of the Laplacian matrix.

In their algorithm, X is a 0-1 indicator matrix (that is, $c_{\text{yes}} = 1$). Not every 0-1 matrix represents an acceptable partition, so they impose two linear constraints: a *row-sum constraint* and a *balance constraint*, which one could also call the *column-sum constraint*. The row-sum constraint dictates that the indicator matrix assigns each vertex to one and only one subgraph. It is written

$$X\mathbf{1}_k = \mathbf{1}_n,$$

where $\mathbf{1}_k$ denotes the vector of k 1's.

The balance constraint is a k-subgraph analog of the constraint (22); it dictates that each subgraph has the correct mass. Let $\mu_1 \geq \mu_2 \geq \ldots \geq \mu_k$ be the desired masses of the subgraphs, with $\sum_{i=1}^{k} \mu_i = \text{Mass}(G)$.

Define a diagonal $k \times k$ *subgraph mass matrix* $\mathbb{M}$ whose diagonal entries are $\mathbb{M}_{ii} = \mu_i = \text{Mass}(G_i)$, arranged in decreasing order. The balance constraint is a constraint on the column sums of MX, written

$$\mathbf{1}_n^\mathsf{T} MX = \mathbf{1}_k^\mathsf{T} \mathbb{M}.$$

Graph partitioning is thus equivalent to the discrete optimization problem of choosing X to

$$\begin{aligned} \text{minimize} \quad & \text{trace}\left(X^\mathsf{T} L X\right) \\ \text{subject to} \quad & \forall i, j, X_{ji} = 0 \text{ or } X_{ji} = 1 \\ \text{and} \quad & \mathbf{1}_n^\mathsf{T} MX = \mathbf{1}_k^\mathsf{T} \mathbb{M} \\ \text{and} \quad & X\mathbf{1}_k = \mathbf{1}_n. \end{aligned} \tag{41}$$

Relax the problem by replacing the discrete constraint with the quadratic constraint

$$X^\mathsf{T} MX = \mathbb{M}.$$

This identity encodes one constraint for each entry of $\mathbb{M}$. For each diagonal entry $\mathbb{M}_{ii} = \mu_i$, the relaxed problem constrains the corresponding column vector X_{*i} to lie on an ellipsoid that is centered at the origin and passes through every 0-1 point signifying a total mass of μ_i. (If every vertex has mass 1, the ellipsoid is a sphere of radius $\sqrt{\mu_i}$.) In the absence of the discrete 0-1 constraint, these ellipsoidal constraints help by preventing X from having very large components.

For each off-diagonal entry $\mathbb{M}_{ij} = 0$, the relaxed problem constrains the column vectors X_{*i} and X_{*j} to be M-orthogonal to each other; observe that a 0-1 indicator matrix that satisfies the row-sum constraint will always have M-orthogonal columns. We are relaxing the 0-1 constraint, but the M-orthogonality constraint and the row-sum constraint together help to nudge the continuous optimum a little closer to a 0-1 matrix. The most important observation is that every 0-1 matrix that satisfies the row-sum and balance constraints also satisfies the quadratic constraint; our introducing the quadratic constraint does not rule out the solution of the discrete problem (41) nor any approximate solutions.

It is instructive to consider the relaxed optimization problem if we drop the linear constraints. Then the simplified problem is to

$$\begin{aligned} \text{minimize} \quad & \text{trace}\left(X^\mathsf{T} L X\right) \\ \text{subject to} \quad & X^\mathsf{T} MX = \mathbb{M}. \end{aligned} \tag{42}$$

Observe the similarity to the relaxed optimization problem (13) for two-way partitioning. It should come as no surprise that the optimal X has as its column vectors the first k eigenvectors, scaled appropriately. This solution is not very useful for partitioning, not least because the components of the eigenvectors (except v_1) are as likely to be negative as positive, so X is not close to any 0-1 indicator matrix. But this solution will give us a simple lower bound on the size of the optimal cut, along with some observations that will help us solve the optimization problem when we reintroduce the linear constraints. Moreover, we will use the solution in a slightly different way in Section E.4.

Let $\lambda_1 = 0 \leq \lambda_2 \leq \ldots \leq \lambda_k$ be the k least eigenvalues of the generalized eigensystem $Lv = \lambda Mv$ (including multiplicities), and let $v_1 \propto \mathbf{1}, v_2, \ldots, v_k$ be the corresponding eigenvectors. Scale the eigenvectors so that $v_i^\mathsf{T} M v_i = 1$ and collect them in an $n \times k$ matrix $V = [v_1, \ldots, v_k]$. The generalized eigenvectors are mutually M-orthogonal, so $V^\mathsf{T} MV = I$. Let Λ be the diagonal $k \times k$ matrix whose diagonal entries are the eigenvalues in increasing order. As $Lv_i = \lambda_i M v_i$, $LV = MV\Lambda$.

A solution to the relaxed optimization problem (42) is

$$X = V\mathbb{M}^{1/2},$$

for which the objective value is

$$\begin{aligned} \operatorname{trace}\left(\mathbb{M}^{1/2}V^{\mathsf{T}}LV\mathbb{M}^{1/2}\right) &= \operatorname{trace}\left(V^{\mathsf{T}}MV\Lambda\mathbb{M}\right) \\ &= \operatorname{trace}(\Lambda\mathbb{M}) \\ &= \sum_{i=1}^{k} \lambda_i\mu_i. \end{aligned}$$

We can drop the first term of the summation, as $\lambda_1 = 0$. Note that it is important that the masses are in decreasing order, whereas the eigenvalues are in increasing order, and $\mathbb{M}$, Λ, and V are ordered accordingly. Intuitively, the solution is trying to match the sparsest cuts of G with the largest subgraph masses.

The original discrete optimization problem (41) cannot have a minimum objective value less than the minimum for the less-constrained problem (42). It follows from Equation (38) that for *every* k-subgraph partition satisfying the balance constraint,

$$\operatorname{Cut}(G_1, G_2, \ldots, G_k) \geq \frac{1}{2}\sum_{i=2}^{k} \lambda_i\mu_i. \tag{43}$$

Donath and Hoffman [18] proved this inequality in 1973 for the special case where every edge weight and vertex mass is 1, using different methods. The derivation given here, with its extension to weighted graphs, comes from Rendl and Wolkowicz [60].

To obtain a practical partitioning algorithm and a stronger lower bound, let us reintroduce the two linear constraints of (41). Rendl and Wolkowicz explicitly enforce the row-sum constraint by projecting X linearly into the affine subspace of 0-1 matrices whose rows satisfy the constraint. You might imagine simply eliminating the variables in one column of X, as they are determined by the other columns and the row-sum constraint. However, Rendl and Wolkowicz suggest a more clever projection that will help us to enforce the balance constraint easily.

Let R be a $k \times k$ matrix whose first column is proportional to $\mathbf{1}_k$ and whose columns are mutually $\mathbb{M}$-orthogonal and scaled so that $R^{\mathsf{T}}\mathbb{M}R = I$. We write

$$R = \begin{bmatrix} \frac{1}{\sqrt{\operatorname{Mass}(G)}}\mathbf{1}_k & Q \end{bmatrix},$$

where the columns of $Q \in \mathbb{R}^{k\times(k-1)}$ are a basis for the subspace $\mathbb{M}$-orthogonal to $\mathbf{1}_k$. Any mutually $\mathbb{M}$-orthogonal basis for that subspace will do; one way to compute a suitable Q is by Gram–Schmidt conjugation [27, Section 5.2.8] applied to $\mathbf{1}_k$ and the Cartesian axes for $\mathbb{R}^k$. Observe that $Q^{\mathsf{T}}\mathbb{M}Q = I$ and $\mathbf{1}_k^{\mathsf{T}}\mathbb{M}Q = \mathbf{0}$.

If a matrix X satisfies the row-sum constraint $X\mathbf{1}_k = \mathbf{1}_n$, then

$$XR = \begin{bmatrix} \frac{1}{\sqrt{\operatorname{Mass}(G)}}\mathbf{1}_n & XQ \end{bmatrix} \equiv \begin{bmatrix} \frac{1}{\sqrt{\operatorname{Mass}(G)}}\mathbf{1}_n & Y \end{bmatrix},$$

where $Y = XQ \in \mathbb{R}^{n\times(k-1)}$ is a projection of X onto a smaller subspace. The idea is that permitting X to vary over matrices that satisfy the row-sum constraint is equivalent to permitting Y to vary over all $n \times (k-1)$ matrices, with no constraint.

If we can find a suitable Y, we can reconstruct a corresponding X that is guaranteed to satisfy the row-sum constraint by taking advantage of the $\mathbb{M}$-orthogonality of R.

$$X = XRR^{-1} = (XR)R^{\mathsf{T}}\mathbb{M} = \frac{1}{\text{Mass}(G)}\mathbf{1}_n\mathbf{1}_k^{\mathsf{T}}\mathbb{M} + YQ^{\mathsf{T}}\mathbb{M}. \tag{44}$$

The first term on the right-hand side is a matrix that satisfies both the row-sum constraint and the balance constraint. The second term enables X to take on any value within the subspace of matrices that satisfy the row-sum constraint. Moreover, if the columns of Y happen to be M-orthogonal to $\mathbf{1}$, then X will satisfy the balance constraint. The entries of Y are degrees of freedom that specify linear combinations of rows of $Q^{\mathsf{T}}\mathbb{M}$ that are added to rows of X.

Let us rewrite the continuous optimization problem in terms of Y. We can drop the row-sum constraint, because the right-hand side of (44) satisfies it for every Y. The balance constraint, $\mathbf{1}_n^{\mathsf{T}}MX = \mathbf{1}_k^{\mathsf{T}}\mathbb{M}$, implies that $\mathbf{1}_n^{\mathsf{T}}MY = \mathbf{1}_n^{\mathsf{T}}MXQ = \mathbf{1}_k^{\mathsf{T}}\mathbb{M}Q = \mathbf{0}$. The quadratic constraint, $X^{\mathsf{T}}MX = \mathbb{M}$, implies that $Y^{\mathsf{T}}MY = Q^{\mathsf{T}}X^{\mathsf{T}}MXQ = Q^{\mathsf{T}}\mathbb{M}Q = I$. From (44), one can show that the converse implications also hold. Recall that $L\mathbf{1}_n = 0$; it follows from (44) that $LX = LYQ^{\mathsf{T}}\mathbb{M}$ and the projected problem is to choose Y to

$$\begin{aligned} \text{minimize} \quad & \text{trace}\left(X^{\mathsf{T}}LX\right) = \text{trace}\left(\mathbb{M}QY^{\mathsf{T}}LYQ^{\mathsf{T}}\mathbb{M}\right) = \text{trace}\left(Q^{\mathsf{T}}\mathbb{M}^2QY^{\mathsf{T}}LY\right) \\ \text{subject to} \quad & Y^{\mathsf{T}}MY = I \\ \text{and} \quad & \mathbf{1}_n^{\mathsf{T}}MY = \mathbf{0}^{\mathsf{T}}. \end{aligned} \tag{45}$$

The balance constraint dictates that every column of Y be M-orthogonal to $\mathbf{1}$. Observe that this constraint (unlike the balance constraint on X) has the same form as our usual balance constraint (12). We will see that the optimal solution expresses the columns of Y as linear combinations of the eigenvectors $v_2, \ldots, v_k$; the balance constraint will be enforced for free because these eigenvectors are M-orthogonal to $v_1 \propto \mathbf{1}_n$.

Let $\dot{V} = [v_2, \ldots, v_k]$ be the $n \times (k-1)$ matrix obtained by deleting v_1 from V, and let $\dot{\Lambda}$ be the diagonal $(k-1) \times (k-1)$ matrix obtained by deleting λ_1 and its row and column from Λ. We still have $L\dot{V} = M\dot{V}\dot{\Lambda}$ and $\dot{V}^{\mathsf{T}}M\dot{V} = I$.

Let $\hat{\mu}_1 \geq \hat{\mu}_2 \geq \ldots \hat{\mu}_{k-1}$ be the eigenvalues of the matrix $Q^{\mathsf{T}}\mathbb{M}^2Q$; note that they are indexed in *descending* order. Let $\hat{\Lambda}$ be the diagonal $(k-1) \times (k-1)$ matrix listing these eigenvalues in descending order. Let $\hat{V}$ be the $(k-1) \times (k-1)$ matrix whose columns are the corresponding *unit* eigenvectors, ordered by decreasing eigenvalue. Thus $Q^{\mathsf{T}}\mathbb{M}^2Q\hat{V} = \hat{V}\hat{\Lambda}$ and $\hat{V}^{\mathsf{T}}\hat{V} = I$. Note that it is important that the columns of $\dot{V}$ are ordered by increasing eigenvalue, whereas the columns of $\hat{V}$ and $\hat{\Lambda}$ are ordered by decreasing eigenvalue.

A solution to the optimization problem (45) is

$$Y = \dot{V}\hat{V}^{\mathsf{T}},$$

for which the objective function is

$$\begin{aligned} \text{trace}\left(Q^{\mathsf{T}}\mathbb{M}^2QY^{\mathsf{T}}LY\right) &= \text{trace}\left(Q^{\mathsf{T}}\mathbb{M}^2Q\hat{V}\dot{V}^{\mathsf{T}}L\dot{V}\hat{V}^{\mathsf{T}}\right) \\ &= \text{trace}\left(\hat{V}\hat{\Lambda}\dot{V}^{\mathsf{T}}M\dot{V}\dot{\Lambda}\hat{V}^{\mathsf{T}}\right) \\ &= \text{trace}\left(\dot{\Lambda}\hat{V}^{\mathsf{T}}\hat{V}\hat{\Lambda}\right) \\ &= \text{trace}(\dot{\Lambda}\hat{\Lambda}) \\ &= \sum_{i=1}^{k-1} \lambda_{i+1}\hat{\mu}_i. \end{aligned}$$

This solution gives us a lower bound, similar to (43) but stronger, on the weight of every k-subgraph partition satisfying the balance constraint,

$$\mathrm{Cut}(G_1, G_2, \ldots, G_k) \geq \frac{1}{2} \sum_{i=1}^{k-1} \lambda_{i+1} \hat{\mu}_i.$$

This bound rederives the left-hand side of Inequality (20) for $k = 2$ (for which $\hat{\mu}_1 = 2\mu_1\mu_2/(\mu_1 + \mu_2)$). For $k = 3$, let $\mu = \mathrm{Mass}(G) = \mu_1 + \mu_2 + \mu_3$. The values of $\hat{\mu}_1$ and $\hat{\mu}_2$ are

$$\frac{\mu_1\mu_2 + \mu_1\mu_3 + \mu_2\mu_3 \pm \sqrt{\mu_1^2\mu_2^2 + \mu_1^2\mu_3^2 + \mu_2^2\mu_3^2 - \mu_1\mu_2\mu_3\mu}}{\mu},$$

with addition yielding $\hat{\mu}_1$ and subtraction yielding $\hat{\mu}_2$. Therefore, every three-way cut with subgraph masses μ_1, μ_2, and μ_3 satisfies

$$\mathrm{Cut}(G_1, G_2, G_3) \geq \frac{(\lambda_2 + \lambda_3)(\mu_1\mu_2 + \mu_1\mu_3 + \mu_2\mu_3) - (\lambda_3 - \lambda_2)\sqrt{\mu_1^2\mu_2^2 + \mu_1^2\mu_3^2 + \mu_2^2\mu_3^2 - \mu_1\mu_2\mu_3\mu}}{2\mu}.$$

Let us review the multipartitioning algorithm so far. Begin by computing the eigenvectors $v_2, \ldots, v_k$ of the generalized eigensystem $Lv = \lambda Mv$, scale them so that $v_i^\mathsf{T} M v_i = I$, and collect them in $\dot{V}$, ordered by increasing eigenvalues. Compute a projection matrix Q, probably by Gram–Schmidt conjugation. Compute all $k - 1$ unit eigenvectors of the matrix $Q^\mathsf{T}\mathbb{M}^2 Q$ and collect them in $\hat{V}$, ordered by decreasing eigenvalues. Compute $X = (1/\mathrm{Mass}(G))\mathbf{1}_n\mathbf{1}_k^\mathsf{T}\mathbb{M} + \dot{V}\hat{V}^\mathsf{T}Q^\mathsf{T}\mathbb{M}$.

It remains only to round the continuous solution X to a 0-1 indicator matrix. Unfortunately, Rendl and Wolkowicz do not specify a rounding method except for $k = 2$, in which case they simply cut the Fiedler vector at the spot that gives the correct subgraph masses. For $k \geq 3$, rounding an indicator matrix is more complicated than rounding an indicator vector.

Fortunately, Hendrickson and Leland [33] have a suggestion. They cast the problem of rounding X as a *minimum cost assignment problem*, which uses the following model. For each pair (i, j), where $i \in [1, n]$ is a vertex number and $j \in [1, k]$ is a subgraph number, assign some real-valued cost $\omega(i, j)$ to the act of placing vertex i in subgraph G_j. The problem is to place μ_j vertices in each subgraph G_j while minimizing the total cost of doing so. Hendrickson and Leland solve this problem with an expected $O(kn + k^{2.5}n^{0.5}\log^{1.5} n)$-time algorithm of Tokuyama and Nakano [67]. Note that this running time is linear in the number n of vertices, and therefore fast if the number k of subgraphs is small.

The choice of costs is heuristic. A straightforward choice is to set $\omega(i, j)$ to be the amount by which $\mathrm{trace}(X^\mathsf{T}LX)$ would increase if row i of X were changed to zeros with one 1 in column j. Of course, this cost model is imperfect because it does not model interactions between changes in different rows of X—but that is as we should expect, as we are trying to solve an NP-hard problem. The model is reasonably accurate if the changes are small.

E.4 Hypercube Partitions and Eigenvector Rotation

Hendrickson and Leland [33] exploit the orthogonality of the Laplacian's eigenvectors so that k eigenvectors induce 2^k subgraphs of equal size (where size is measured in vertices; their algorithm does not seem to extend easily to vertex masses). Their algorithm works best for quadrisection ($k = 2$) or octasection ($k = 3$), either of which can be applied recursively.

The algorithm employs a signed indicator matrix X (instead of a 0-1 indicator matrix); each component is 1 or -1. Unlike in the formulation of Rendl and Wolkowicz, X does not have one column for each subgraph; instead, its k columns distinguish among 2^k subgraphs, each column supplying a one-bit label to each vertex. Think of row i of X as a base-2 number that specifies which subgraph vertex i belongs to. For lack of another name, I call the output a *hypercube partition*—primarily because a hypercube has edges in k orthogonal directions meeting at 2^k vertices, but also because Hendrickson and Leland advocate their approach for mapping parallel computations to hypercube communication architectures.

Consider the discrete optimization problem of choosing X to

$$\begin{aligned} \text{minimize} \quad & \sum_{i=1}^{k} X_{*i}^{\mathsf{T}} L X_{*i} \\ \text{subject to} \quad & \forall i, \forall j, X_{ji} = 1 \text{ or } X_{ji} = -1 \\ \text{and} \quad & \forall i, \mathbf{1}^{\mathsf{T}} X_{*i} = 0 \\ \text{and} \quad & \forall i \neq j, X_{*i}^{\mathsf{T}} X_{*j} = 0. \end{aligned} \tag{46}$$

The middle constraint requires that each indicator vector have equally many 1's as -1's. The final constraint requires that each pair of indicator vectors agree and disagree on equally many vertices. For $k = 2$, the last two constraints ensure that all four subgraphs have equally many vertices. For $k = 3$, the constraints do not quite suffice to balance all eight subgraphs; an added constraint $\sum_i X_{i1} X_{i2} X_{i3} = 0$ (on the third moment of the distribution) would suffice. For $k > 3$, the number of constraints (on higher moments) increases exponentially as k increases.

The objective function (46) looks like the function (38), but its interpretation is different; because of the base-2 subgraph encoding, it counts some edges multiple times. If the two vertices incident on an edge lie in different subgraphs, the edge's weight is counted once for each bit in which the subgraph encodings differ. Hendrickson and Leland argue that this is appropriate for computations on a hypercube communication architecture where the communication cost is proportional to the number of links traversed. (For most other applications, the extra counting probably does little harm.)

Relax the discrete optimization problem to a continuous optimization problem by relaxing the first constraint to $X_{*i}^{\mathsf{T}} X_{*i} = n$. The relaxed optimization problem is to choose X to

$$\begin{aligned} \text{minimize} \quad & \operatorname{trace}\left(X^{\mathsf{T}} L X\right) \\ \text{subject to} \quad & X^{\mathsf{T}} X = nI \\ \text{and} \quad & \mathbf{1}^{\mathsf{T}} X = \mathbf{0}. \end{aligned}$$

Here, the constraint $X^{\mathsf{T}} X = nI$ expresses simultaneously the constraints $X_{*i}^{\mathsf{T}} X_{*i} = n$ and $X_{*i}^{\mathsf{T}} X_{*j} = 0$.

A solution of the relaxed optimization problem is $V = [v_2, \ldots, v_{k+1}]$, where $v_2, \ldots, v_{k+1}$ are the eigenvectors of the Laplacian matrix L, each scaled to have length $\sqrt{n}$. The minimum objective value is $\operatorname{trace}(V^{\mathsf{T}} L V) = \sum_{i=2}^{k+1} v_i^{\mathsf{T}} L v_i = n \sum_{i=2}^{k+1} \lambda_i$.

But V is not the only solution. For any solution V and any orthonormal matrix $R \in \mathbb{R}^{k \times k}$, VR is also a solution. Recall that a square matrix R is orthonormal if $R^{\mathsf{T}} R = I = R R^{\mathsf{T}}$; an orthonormal R has the effect of rotating (or reflecting) the rows of V (the binary labels identifying the subgraphs) together around the origin in k-dimensional space. Hendrickson and Leland exploit this observation by finding a rotation matrix R that produces an indicator matrix VR closer to the ideal where every component is ± 1.

Let us verify the claim that if V is a solution to the relaxed optimization problem, then so is VR. First, observe from (39) that the objective function is $\operatorname{trace}(R^{\mathsf{T}} V^{\mathsf{T}} L V R) = \operatorname{trace}(V^{\mathsf{T}} L V R R^{\mathsf{T}}) = \operatorname{trace}(V^{\mathsf{T}} L V)$, so

its value is unchanged by the rotation. Second, observe that the constraints $R^\mathsf{T}V^\mathsf{T}VR = R^\mathsf{T}nIR = nI$ and $\mathbf{1}^\mathsf{T}VR = \mathbf{0}$ are still satisfied. Intuitively, rotating the rows of the indicator matrix in $\mathbb{R}^k$ changes nothing essential about a solution to the relaxed optimization problem.

See the article by Hendrickson and Leland [33] for their method for choosing R to yield an indicator matrix $X = VR$ whose components are closer to ± 1. For $k > 2$ they also try to satisfy moment constraints like $\sum_i X_{i1}X_{i2}X_{i3} = 0$ to improve balance (but if $k > 3$, not all the moment constraints can be satisfied). This rotation of the indicator vectors is the main reason the method can produce better partitions than recursive spectral bisection. (See Section E.5 for an alternative eigenvector rotation algorithm, designed for clustering rather than balanced subgraphs.)

Although the balance constraints help to produce subgraphs of equal size, this goal is really achieved by the rounding method. Hendrickson and Leland round the indicator matrix to have discrete ± 1 values as described in Section E.3: they model the rounding as a minimum cost assignment problem [67] that assigns $n/2^k$ of the vertices to each of the discrete points $(\pm 1, \ldots, \pm 1)$. They use a simpler (and probably less effective) cost model than the one suggested in Section E.3: the cost of assigning a vertex to a discrete point is simply the Euclidean distance from the spectral vector to the discrete point.

Hypercube partitioning is less flexible than vector partitioning, which can produce an arbitrary number of subgraphs, arbitrarily balanced, taking into account differing vertex masses. But it is specialized for perfectly balanced partitions, and it saves computation time by producing more subgraphs from fewer eigenvectors.

E.5 Spectral Clustering by Spectral Vector Rotation (and the Normalized Multicut)

Yu and Shi [70] propose an elegant multipartitioning algorithm that uses the first k eigenvectors of the generalized eigensystem (including $v_1 \propto \mathbf{1}$) to partition a graph into k subgraphs. It was designed for graph clustering, but it adapts easily if the sizes of the subgraphs are specified in advance. Yu and Shi try to minimize the sum of the subgraphs' isoperimetric ratios (in contrast to Rendl and Wolkowicz [60], who try to minimize the total cut weight). They assign masses to vertices as described in Section A, and call the optima "multiclass normalized cuts"; but their method generalizes to arbitrary positive vertex masses. Like Hendrickson and Leland [33], Yu and Shi round a continuous solution matrix by rotating its rows to bring it closer to a discrete indicator matrix; but their objective function and their interpretation of the indicator matrix are different. A notable feature of their algorithm is how it chooses a rotation that minimizes the distance of the solution matrix from some indicator matrix.

Let Z be an $n \times k$ indicator matrix such that Z_{ji} is a specified value $c_i \neq 0$ if vertex j is in subgraph G_i, and zero otherwise. The goal is to minimize the objective function

$$\sum_{i=1}^{k} \frac{\text{Cut}(G_i, G\backslash G_i)}{\text{Mass}(G_i)} = \sum_{i=1}^{k} \frac{Z_{*i}^\mathsf{T} L Z_{*i}}{Z_{*i}^\mathsf{T} M Z_{*i}}, \tag{47}$$

which is the sum of the subgraphs' isoperimetric ratios, and also is a sum of generalized Rayleigh quotients. The identity (47) holds regardless of the choice of c_i, and each subgraph G_i can employ a different c_i. Although Yu and Shi's algorithm actually computes a 0-1 indicator matrix X, it is easiest to formulate the optimization problem in terms of an indicator matrix Z that uses different values. Recall from Section 2.6 the trick of choosing an indicator value that depends on the mass of the subgraph. Let

$$c_i = \frac{1}{\sqrt{\text{Mass}(G_i)}}.$$

As before, this trick does not require us to know in advance what the mass of G_i will be. With this choice,

$$Z_{*i}^{\mathsf{T}} M Z_{*i} = 1, \tag{48}$$

which simplifies Expression (47) by eliminating the denominators. In other words, c_i is chosen so that $Z_{*i}^{\mathsf{T}} L Z_{*i}$ is the isoperimetric ratio of G_i (rather than the cut weight, as it would be if Z were a 0-1 indicator matrix.) Imposing the constraint (48) simplifies optimizing the objective function (47).

To ensure that no vertex is assigned to more than one subgraph, constrain the subgraphs' indicator vectors to be mutually M-orthogonal. (Ordinary orthogonality would work too, but the eigenvectors will give us M-orthogonality for free.) Therefore, the discrete optimization problem is to

$$\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{k} Z_{*i}^{\mathsf{T}} L Z_{*i} \\
\text{subject to} \quad & \forall i, \forall j, Z_{ji} = 0 \text{ or } Z_{ji} = \frac{1}{\sqrt{\text{Mass}(G_i)}} \\
\text{and} \quad & \forall i, Z_{*i}^{\mathsf{T}} M Z_{*i} = 1 \\
\text{and} \quad & \forall i \neq j, Z_{*i}^{\mathsf{T}} M Z_{*j} = 0.
\end{aligned}$$

Discard the first constraint, producing a continuous, relaxed optimization problem that can be written in the abbreviated form

$$\begin{aligned}
\text{minimize} \quad & \text{trace}\left(Z^{\mathsf{T}} L Z\right) \\
\text{subject to} \quad & Z^{\mathsf{T}} M Z = I,
\end{aligned}$$

which, as promised, does not require us to know $\text{Mass}(G_i)$.

One solution of the relaxed problem[9] is to set Z's columns to be the first k eigenvectors of the generalized eigensystem $Lv = \lambda Mv$—that is, choose Z to be $V = [v_1, \ldots, v_k] \in \mathbb{R}^{n \times k}$—where each eigenvector is scaled so that $v_i^{\mathsf{T}} M v_i = 1$. Unfortunately, this solution makes a lousy indicator matrix, because $v_1 \propto \mathbf{1}$ and the other eigenvectors have as many negative components as positive ones.

Like Hendrickson and Leland [33], Yu and Shi obtain solutions more amenable to rounding by rotating the rows of V. Let $R \in \mathbb{R}^{k \times k}$ be an orthonormal matrix (i.e. a rotation or reflection), so $R^{\mathsf{T}} R = I = R R^{\mathsf{T}}$. Then VR is also a solution of the relaxed problem. Let's see why.

As argued in Section E.4, $\text{trace}(R^{\mathsf{T}} V^{\mathsf{T}} L V R) = \text{trace}(V^{\mathsf{T}} L V)$, so the objective function is invariant under rotation of the rows of V, and therefore it is minimized by VR for any orthonormal R. The constraint also holds for VR: given that $V^{\mathsf{T}} M V = I$, we have $R^{\mathsf{T}} V^{\mathsf{T}} M V R = R^{\mathsf{T}} R = I$. (In other words, after a rotation of the rows of V, the columns are still mutually M-orthogonal and of the correct lengths.) It follows that, subject to the constraint $Z^{\mathsf{T}} M Z = I$, the objective function (47) is also invariant under row rotation.

The goal is to find an orthonormal R such that VR is as close as possible to some valid indicator matrix. Yu and Shi's algorithm for choosing R uses a 0-1 indicator matrix X, constrained so that every row has exactly one nonzero entry (always 1). The matrices X and Z have the same meaning—they both have the same zero components—but only the nonzero components of Z depend on the subgraph masses.

[9] The solution $Z = V$ follows from the continuous optimization problem by writing $\nabla \left(\sum_{i=1}^{k} Z_{*i}^{\mathsf{T}} L Z_{*i} \right) = \sum_{i=1}^{k} \lambda_i \nabla \left(Z_{*i}^{\mathsf{T}} M Z_{*i} \right)$, where each λ_i is a Lagrange multiplier. As $v_i^{\mathsf{T}} L v_i = \lambda_i$, the objective value for this (and every optimal) solution is $\text{trace}(V^{\mathsf{T}} L V) = \sum_{i=1}^{k} \lambda_i$.

Define the diagonal $k \times k$ *subgraph mass matrix*

$$\mathbb{M} = X^\mathsf{T} M X$$

whose diagonal entries are $\mathbb{M}_{ii} = \text{Mass}(G_i)$. Then the two indicator matrices are related by

$$X = Z\mathbb{M}^{1/2}.$$

Observe that for any continuous Z that minimizes (47) and any diagonal $\mathbb{M}$, $Z\mathbb{M}^{1/2}$ also minimizes (47). Therefore, $VR\mathbb{M}^{1/2}$ minimizes (47) for any orthonormal R.

Now the goal is to find a rotation R such that $VR\mathbb{M}^{1/2}$ is as close as possible to some 0-1 indicator matrix. With V fixed, consider the problem of finding a 0-1 indicator matrix X and an orthonormal rotation R that minimizes the mass-weighted sum-of-squares rounding error

$$\sum_{i=1}^{k} \sum_{j=1}^{n} m_j (X - VR\mathbb{M}^{1/2})_{ji}^2. \tag{49}$$

Yu and Shi iteratively find a local optimum[10] by alternately solving for R (with X fixed) and solving for X (with R fixed) until the objective function (49) stops decreasing—or perhaps more importantly, until the objective function (47) stops decreasing. To start the iterations, Yu and Shi recommend generating an initial 0-1 indicator matrix X with the algorithm of Ng, Jordan, and Weiss [54] described in Section E.2, although they note that the iteration is robust to an arbitrary starting X or R.

For a fixed R (and $\mathbb{M}$), an optimal 0-1 matrix X is easily found by rounding. For each row j of $VR\mathbb{M}^{1/2}$, simply find the column i with the greatest component, and assign vertex j to subgraph G_i. Once X is determined, update $\mathbb{M} = X^\mathsf{T} M X$.

For a fixed X, finding an optimal rotation R takes more effort. Observe that Expression (49) is equal to

$$\begin{aligned} & \operatorname{trace}\left((X - VR\mathbb{M}^{1/2})^\mathsf{T} M (X - VR\mathbb{M}^{1/2})\right) \\ = \; & \operatorname{trace}(X^\mathsf{T} M X) + \operatorname{trace}(\mathbb{M}^{1/2} R^\mathsf{T} V^\mathsf{T} M V R \mathbb{M}^{1/2}) - 2\operatorname{trace}(X^\mathsf{T} M V R \mathbb{M}^{1/2}) \\ = \; & 2\,\text{Mass}(G) - 2\operatorname{trace}(\mathbb{M}^{1/2} X^\mathsf{T} M V R). \end{aligned}$$

Thus minimizing the rounding error (49) is equivalent to maximizing the trace of $\mathbb{M}^{1/2} X^\mathsf{T} M V R$. To find the rotation R that does so, compute the singular value decomposition

$$U \Sigma W^\mathsf{T} = \mathbb{M}^{1/2} X^\mathsf{T} M V,$$

where $U, W \in \mathbb{R}^{k \times k}$ are orthonormal matrices, and $\Sigma \in \mathbb{R}^{k \times k}$ is a diagonal matrix of nonnegative singular values. Then the optimal rotation is

$$R = W U^\mathsf{T},$$

[10] I have taken the liberty of presenting an objective function (49) that differs in two ways from the one suggested by Yu and Shi. First, Yu and Shi do not use vertex masses to weight the errors in Expression (49). I include the term m_j to ensure that a vertex of mass 2 is treated similarly to two vertices of mass 1 that have the same neighbors. Second, Yu and Shi do not compute $\mathbb{M}$; instead, they scale the rows of V to have unit length. If every row of VR happens to have exactly one nonzero component, this scaling yields a 0-1 indicator matrix, and therefore achieves the same result as scaling the columns of VR by $\mathbb{M}^{1/2}$. Otherwise, this scaling sacrifices the property of minimizing (47) even before rounding, so I present here a slightly different procedure that sidesteps this unnecessary loss of optimality. To obtain Yu and Shi's original algorithm, cross out M and $\mathbb{M}$ from the rest of the presentation, and normalize the rows of V to unit length to approximate the effect of $\mathbb{M}^{1/2}$.

for which the objective value is $\text{trace}(\mathbb{M}^{1/2}X^\mathsf{T}MVR) = \text{trace}(U\Sigma W^\mathsf{T}WU^\mathsf{T}) = \text{trace}(\Sigma)$, which is the sum of the singular values.

To summarize the algorithm: compute the first k eigenvectors of the generalized eigensystem $Lv = \lambda Mv$ and collect them as the columns of a matrix V, scaled so that $V^\mathsf{T}MV = I$. Guess an initial 0-1 indicator matrix X, perhaps with the algorithm of Ng et al. (Section E.2). Iterate the following steps: compute the subgraph mass matrix $\mathbb{M} = X^\mathsf{T}MX$; compute the singular value decomposition $U\Sigma W^\mathsf{T} = \mathbb{M}^{1/2}X^\mathsf{T}MV$; compute $R = WU^\mathsf{T}$; and compute a new X by rounding $VR\mathbb{M}^{1/2}$ to a 0-1 indicator matrix (by selecting the greatest component in each row). Repeat until the objective function (47) or (49) stops decreasing.

Yu and Shi present their algorithm as a graph clustering algorithm—the subgraph masses are not prescribed in advance. However, if the user wishes to prescribe the subgraph masses, two changes seem sufficient. First, the subgraph mass matrix $\mathbb{M}$ should be fixed to reflect the desired subgraph masses; it should not change with X. Second, round $VR\mathbb{M}^{1/2}$ to a 0-1 indicator matrix by casting the rounding problem as a minimum cost assignment problem, as described at the end of Section E.3.

E.6 Bounds for k-Subgraph Partitions

Recall that a graph has k connected components if and only if the first k eigenvalues of the Laplacian matrix are all zero. The eigenvalues vary continuously as the weight of an edge changes, so it seems intuitively reasonable that if $\lambda_2, \ldots, \lambda_k$ are nonzero but very small, then the graph probably has a small cut that partitions it into k subgraphs. This intuition seems to be confirmed by researchers in graph clustering, who sometimes employ an "eigengap heuristic" that cuts a graph into k pieces if λ_{k+1} is notably larger than λ_k.

Lee, Oveis Gharan, and Trevisan [46] provide theoretical support for this observation by showing that a graph can be cut into $k \geq 2$ subgraphs such that each subgraph has an isoperimetric ratio in $O(k^2\sqrt{\lambda_k})$. Here, the *isoperimetric ratio* of a single subgraph is the total weight of the cut edges incident on that subgraph divided by the total mass of that subgraph. Moreover, they show that $O(\sqrt{\lambda_{2k}\log k})$ is an alternative upper bound—or better yet, $O(\sqrt{\lambda_{2k}})$ if the graph is planar. The authors call these bounds "multi-way Cheeger inequalities"; their proofs share features of the proof in Section B.3. They give a partitioning algorithm that obtains these bounds; it is omitted here.

These results apply to problems in graph clustering, but they do not apply if the subgraph masses are specified in advance; a graph might have a good cut into k subgraphs of the wrong sizes. However, recall from Section E.3 that there is a lower bound on the cut weight for prescribed subgraph masses. If G is partitioned into k subgraphs having $\mu_1 \geq \mu_2 \geq \ldots \geq \mu_k$ vertices, then the cut weight is at least $\sum_{i=2}^{k} \lambda_i\mu_i/2$. Moreover, the cut weight is at least $\sum_{i=2}^{k} \lambda_i\hat{\mu}_i/2$, where $\hat{\mu}_i$ is defined in Section E.3.

F Fixed Vertex Values, Iterative Improvement, and Explicit Balancing

Some applications fix some of the vertices to lie in a particular subgraph, thereby biasing the assignments of the unconstrained vertices. Hendrickson, Leland, and Van Driessche [34] describe several uses of this idea, including circuit partitioning where some circuit elements must connect to fixed terminals (such as microchip pins) and biasing recursive bisection so that subgraphs that are connected by many edges are more likely to be assigned to closely connected processors. Fixed vertices are also used by branch-and-bound search algorithms for graph partitioning, which can fix some vertex values and use spectral or isoperimetric partitioning to find a good assignment for the remaining vertices.

If there is time enough to run a partitioner several times, there are several ways to adjust the partitioner on subsequent runs in hopes of finding a better cut. One way is to suppose that a vertex assigned an extreme value is probably labeled well, whereas a vertex near the median deserves further investigation. We can exploit this by assigning some of the more polarized vertices to subgraphs and running the partitioner a second time to label the ambiguous vertices. Because some of the vertices are fixed at their final discrete values, the other vertices have a more accurate model of their effects on the cut weight. This procedure can be iterated several times, each time fixing more of the vertices.

Fixing vertex values fits naturally into isoperimetric partitioning, which always fixes at least one vertex's value. It is trickier to incorporate fixed vertices into spectral partitioning, because they require the solution of an *extended eigenvalue problem* of the form $Lx = \lambda Mx + b + \lambda b'$.

Another iterative improvement method for spectral partitioning begins with the observation that you can change the diagonal of the Laplacian matrix L arbitrarily without changing the discrete minimum of the objective function $x^\mathsf{T} Lx$. A suitable choice of diagonal can force any signed indicator vector to be an eigenvector of the updated system. If the associated eigenvalue is the smallest eigenvalue of the new system, the vector represents a minimum cut. If it is not, we may have an opportunity to find a better cut.

Fixing vertices and changing the diagonal share a pitfall: usually $\mathbf{1}$ is no longer an eigenvector of the updated system, so the balance constraint must be enforced explicitly. Sections F.1 and F.2 include a method for explicitly enforcing the balance constraint, which is also needed in Section G.

F.1 Fixing Vertices in Isoperimetric Partitioning

Consider isoperimetric partitioning when the values of vertices $j+1$ through n are fixed constants, specified in a vector c of length $n - j$. (Typically the components of c are zeros and ones, but they don't have to be.) The partitioner minimizes $x^\mathsf{T} Lx$ subject to the balance constraint $\mathbf{1}^\mathsf{T} Mx = \mu$ and one constraint for each fixed vertex. Let e_i be the unit vector on the x_i-axis. (Component i of e_i is 1; the other components are 0.) A necessary condition for x to be a solution to the constrained optimization problem is

$$\begin{aligned} \nabla(x^\mathsf{T} Lx) &= \beta \nabla(\mathbf{1}^\mathsf{T} Mx) + \sum_{i=j+1}^{n} \gamma_i \nabla x_i, \qquad \text{and therefore} \\ 2Lx &= \beta M\mathbf{1} + \sum_{i=j+1}^{n} \gamma_i e_i, \end{aligned} \tag{50}$$

where β and each γ_i are Lagrange multipliers. The multipliers γ_i imply that we need to satisfy only the first j rows of the linear system $2Lx = \beta M\mathbf{1}$.

In Section 3.3, we saw a method for solving this system that works only if every component of c is zero. Nonzero components make the system harder to solve, because it is no longer easy to factor out β. To eliminate β from the system, use the balance constraint to eliminate x_j. Write the linear system (50) as

$$2\begin{bmatrix} L_{xx} & L_{xj} & L_{xc} \\ L_{xj}^\mathsf{T} & L_{jj} & L_{jc} \\ L_{xc}^\mathsf{T} & L_{jc}^\mathsf{T} & L_{cc} \end{bmatrix}\begin{bmatrix} \tilde{x} \\ x_j \\ c \end{bmatrix} = \beta\begin{bmatrix} M_{xx} & 0 & 0 \\ 0 & M_{jj} & 0 \\ 0 & 0 & M_{cc} \end{bmatrix}\mathbf{1} + \sum_{i=j+1}^{n} \gamma_i e_i \tag{51}$$

where x_j is the jth component of x, and $\tilde{x}$ is a vector of the first $j-1$ components.

Write the balance constraint $\mathbf{1}^\mathsf{T} M x = \mu$ as

$$x_j = \frac{\mu - \mathbf{1}^\mathsf{T} M_{cc} c - \mathbf{1}^\mathsf{T} M_{xx} \tilde{x}}{M_{jj}}. \tag{52}$$

Substituting this expression into the jth row of the system (51) gives

$$\beta = 2\frac{L_{xj}^\mathsf{T}\tilde{x} + L_{jc}c}{M_{jj}} + 2L_{jj}\frac{\mu - \mathbf{1}^\mathsf{T} M_{cc} c - \mathbf{1}^\mathsf{T} M_{xx}\tilde{x}}{M_{jj}^2}. \tag{53}$$

Substituting this expression into the top rows of the system (51) gives the linear system

$$\mathbb{L}\tilde{x} = b \tag{54}$$

where

$$\mathbb{L} = L_{xx} - \frac{1}{M_{jj}}\left(L_{xj}\mathbf{1}^\mathsf{T} M_{xx} + M_{xx}\mathbf{1}L_{xj}^\mathsf{T}\right) + \frac{L_{jj}}{M_{jj}^2} M_{xx}\mathbf{1}\mathbf{1}^\mathsf{T} M_{xx} \tag{55}$$

and

$$b = \left(\frac{L_{jc}c}{M_{jj}} + L_{jj}\frac{\mu - \mathbf{1}^\mathsf{T} M_{cc} c}{M_{jj}^2}\right) M_{xx}\mathbf{1} - \frac{\mu - \mathbf{1}^\mathsf{T} M_{cc} c}{M_{jj}} L_{xj} - L_{xc}c. \tag{56}$$

Solve this system for $\tilde{x}$, then recover x_j from the balance constraint (52).

Observe that $\mathbb{L}$ is dense. If the Laplacian L is sparse, there is no need to explicitly form $\mathbb{L}$. Rather, matrix-vector products of the form $\mathbb{L}z$ can be computed quickly with vector operations, and used in the conjugate gradient method [35] to solve (54).

Incidentially, $\mathbb{L}$ has an intuitive interpretation. The quadratic function $\tilde{x}^\mathsf{T}\mathbb{L}\tilde{x}$ is essentially equal to our objective function $x^\mathsf{T} L x$, but the domain of $\tilde{x}^\mathsf{T}\mathbb{L}\tilde{x}$ is restricted to the intersection of the linear constraints (including the balance constraint). For example, Figure 3 is an illustration of the isocontours of $\tilde{x}^\mathsf{T}\mathbb{L}\tilde{x}$.

F.2 Fixing Vertices in Spectral Partitioning

Consider spectral partitioning when the values of vertices $j+1$ through n are fixed constants, specified in a vector c. (Typically the components of c are ± 1, but they don't have to be.) Unconstrained spectral partitioning has the property that the eigenvectors enforce the balance condition implicitly, but when vertices are fixed, this property must be enforced explicitly as in Section F.1.

Spectral partitioning with fixed vertices has all the constraints of isoperimetric partitioning, plus an additional quadratic constraint $x^\mathsf{T} M x = \text{Mass}(G)$. Solutions to the constrained optimization problem satisfy

$$\begin{aligned} \nabla(x^\mathsf{T} L x) &= \lambda \nabla(x^\mathsf{T} M x) + \beta \nabla(\mathbf{1}^\mathsf{T} M x) + \sum_{i=j+1}^{n} \gamma_i \nabla x_i, \qquad \text{and therefore} \\ 2Lx &= 2\lambda M x + \beta M \mathbf{1} + \sum_{i=j+1}^{n} \gamma_i e_i. \end{aligned}$$

Observe that this is the same condition as Equation (50) with L replaced by $L - \lambda M$. Making this substitution in the linear system (54) yields

$$\mathbb{L}\tilde{x} = \lambda \mathbb{M}\tilde{x} + b + \lambda b' \tag{57}$$

where $\mathbb{L}$ and b are defined in Section F.1 (with $\mu = 0$),

$$\mathbb{M} = M_{xx} + \frac{1}{M_{jj}} M_{xx} \mathbf{1}\mathbf{1}^\mathsf{T} M_{xx}, \tag{58}$$

and

$$b' = \frac{\mathbf{1}^\mathsf{T} M_{cc} c}{M_{jj}} M_{xx} \mathbf{1}.$$

Note that $\mathbb{M}$ is not a diagonal matrix; $\mathbb{M}$ is dense.

Equation (57) is an example of an *extended eigenvalue problem*. Because the balance constraint is explicitly enforced by substitution, we seek the extended eigenvector with the smallest (not second-smallest) eigenvalue λ_1. The extended eigenproblem can be solved by a variant of the Lanczos algorithm [28], although the variation adds more complexity to an already difficult algorithm. Let $\tilde{x}$ be the eigenvector associated with λ_1 in the extended eigensystem (57), then recover x_j from the balance constraint (52).

The method of explicitly enforcing the balance condition shown above is useful in another circumstance: when the diagonal of the Laplacian is changed, as described in Sections F.3 and G.1. Fortunately, if no vertex value is fixed, then $b = b' = \mathbf{0}$ and the problem simplifies to the generalized eigensystem

$$\mathbb{L}\tilde{x} = \lambda \mathbb{M}\tilde{x}. \tag{59}$$

F.3 Changing the Diagonal in Spectral Partitioning

In Section 2.6, we saw that a relaxed problem can be a relaxation of more than one discrete problem. Here we will see that there is more than one way to relax a discrete problem, and different ways yield different continuous solutions.

Consider a discrete optimization problem with a signed indicator vector x (i.e. each component of x is ± 1). For any diagonal matrix D,

$$x^\mathsf{T} D x = \sum_{i=1}^{n} D_{ii}$$

is constant with respect to changes in x. Therefore, the objective function

$$x^\mathsf{T}(L + D)x$$

has the same discrete minimum as $x^T Lx$. We can choose D at our convenience and replace $x^T Lx$ with $x^T(L+D)x$ in the discrete optimization problem, then relax the problem. (See Section G.2 for the analogous transformation for 0-1 indicator vectors and for isoperimetric partitioning.)

In the relaxed optimization problem, however, $x^T Dx$ is not constant. Its inclusion in the objective function tends to move vertices for which D_{ii}/M_{ii} is great closer to zero, thereby pushing vertices for which D_{ii}/M_{ii} is lesser away from zero. We could choose D for several purposes: to better treat edges with negative edge weights (see Section G.1); to break symmetries in the graph that might have $\lambda_2 \approx \lambda_3$; or to nudge the components of the relaxed solution closer to ± 1 (see Section F.4).

Unless D is some scalar times M, $\mathbf{1}$ is not an eigenvector of the new eigensystem $(L + D)v = \lambda Mv$. Therefore, the eigenvectors no longer satisfy the balance constraint. We must enforce it explicitly. Fortunately, Section F.2 shows us how to do so. Simply replace L with $L + D$ in Expression (55) to determine $\mathbb{L}$, compute $\mathbb{M}$ from Expression (58), let $\tilde{x}$ be the eigenvector of the system $\mathbb{L}\tilde{v} = \lambda\mathbb{M}\tilde{v}$ with smallest eigenvalue (computed as discussed in Section H), and recover x_j from the balance constraint (52).

F.4 Verifying the Optimality of a Rounded Solution

One application of changing the diagonal of the Laplacian is to attempt to verify that a rounded solution is optimal, and perhaps improve it if it isn't.

Let x be a signed (discrete) indicator vector. If x represents an unbalanced cut, use the indicator values (16) discussed in Section 2.6, so both the balance constraint and the quadratic constraint are satisfied. Here, no vertex value is fixed, so we eliminate vertex $j = n$ to enforce the balance constraint, and $\tilde{x}$ is a vector comprising the first $n - 1$ components of x. Let Expressions (55) and (58) define $\mathbb{L}$ and $\mathbb{M}$.

It is easy to modify $\mathbb{L}$ so that $\tilde{x}$ is an eigenvector of the generalized eigensystem $\mathbb{L}\tilde{v} = \lambda\mathbb{M}\tilde{v}$ with eigenvalue zero. (The choice of eigenvalue is irrelevant; a different choice would merely shift the spectrum of eigenvalues accordingly.) Compute the unique diagonal matrix $\mathbb{D}$ that satisfies

$$(\mathbb{L} + \mathbb{D})\tilde{x} = 0,$$

so that $\tilde{x}$ is an eigenvector of $\mathbb{L} + \mathbb{D}$ with eigenvalue zero. (Adding $\mathbb{D}$ to $\mathbb{L}$ is equivalent to adding to L a diagonal matrix D whose first $n - 1$ diagonal entries are those of $\mathbb{D}$, with $D_{nn} = 0$.)

Next, compute the smallest eigenvalue λ_1 of the updated eigensystem $(\mathbb{L} + \mathbb{D})\tilde{v} = \lambda\mathbb{M}\tilde{v}$. If $\lambda_1 = 0$, then $\tilde{x}$ is an optimal continuous solution, and therefore an optimal discrete solution![11]

However, if $\lambda_1 < 0$, there is a better continuous solution, namely the corresponding eigenvector $\tilde{v}_1$. Rounding that solution might or might not yield a better discrete solution. Augment $\tilde{v}_1$ to an n-component vector v_1 by recovering the nth component with the balance constraint (52). Round the solution v_1, and check whether it is a better solution than x. If so, replace x with the new rounded solution, and iterate again.

As graph partitioning is NP-hard, this procedure will not always find a demonstrably optimal partition. If $\lambda_1 < 0$ but the rounded eigenvector does not represent a better cut than x, then there might or might not be a better discrete solution than x.

[11] Because of floating-point roundoff, it does not actually suffice to test whether $\lambda_1 = 0$. If λ_1 is nonnegative, or if it is negative but very small, then $\tilde{x}$ is optimal or so close to optimal that floating-point arithmetic cannot tell the difference.

G Graphs with Negative Edge Weights

Most graphs that people want to partition have no negative edge weights, but there are exceptions in such pursuits as image segmentation and surface reconstruction. An edge with negative weight indicates a degree of confidence that its two endpoints should probably be in different subgraphs. For instance, if two pixels in a photograph have sharply different colors, they are likely to be in different objects.

The Laplacian L of a graph with negative edges weights might be indefinite, having negative eigenvalues. This does not entirely invalidate the standard spectral partitioning algorithm, but the eigenvector $\mathbf{1}$ no longer has the smallest eigenvalue; we use v_1 to partition instead of v_2. However, an indefinite Laplacian can cause isoperimetric partitioning to fail, because the continuous optimization problem has no minimum.

The indefinite Laplacian is a symptom of a subtle but serious concern. The continuous optimization problem does not treat positive and negative edge weights symmetrically. The objective function $x^\mathsf{T} Lx$ correctly reflects the weight of a *discrete* cut, but it behaves badly for continuous optimization. By Equation (2), the contribution of edge (i, j) to the objective function is $w_{ij}(x_i - x_j)^2$. A positive edge's contribution improves as its endpoints move toward the same value, but progress brings diminishing rewards. A negative edge earns increasing rewards without limit as its endpoints move further and further apart (which is why the Laplacian is indefinite). The following two sections correct this perverse incentive.

G.1 Negative Edge Weights and Spectral Partitioning

Experience shows that spectral partitioning (with a signed indicator vector) often works better when the contribution of a negative edge weight w_{ij} to the objective function is $-w_{ij}(x_i + x_j)^2$. Thus, a negative edge acts like a positive edge with one of its endpoints negated. This expression rewards endpoints with opposite values, but it cannot drop below zero. Consider minimizing the nonnegative objective function

$$\sum_{w_{ij}>0} w_{ij}(x_i - x_j)^2 - \sum_{w_{ij}<0} w_{ij}(x_i + x_j)^2 = x^\mathsf{T}(L + D)x,$$

where D is the diagonal matrix with components

$$D_{ii} = 2 \sum_{w_{ij}<0} |w_{ij}|, \tag{60}$$

thus

$$(L + D)_{ij} = \begin{cases} -w_{ij} & i \neq j, \\ \sum_{k \neq i} |w_{ik}| & i = j. \end{cases}$$

As we saw in Section F.3, changing the diagonal of the Laplacian does not change the minimum of the discrete optimization problem, though it usually changes the minimum of the relaxed optimization problem. If x is a signed indicator vector (with components ± 1), the objective function is

$$x^\mathsf{T}(L + D)x = 4\,\mathrm{Cut}(G_1, G_2) + 4 \sum_{w_{ij}<0} |w_{ij}|.$$

The summation is a constant, so the modified discrete optimization problem still minimizes the cut weight, but the relaxed optimization problem is better behaved than the original one.

A welcome property of the modified Laplacian $L + D$ is that it is positive semidefinite. A minor disadvantage is that if G has at least one negative edge, $\mathbf{1}$ is not an eigenvector of $L + D$, and the balance condition must be enforced explicitly as described in Sections F.2 and F.3.

Several applications have treated negative edge weights by modifying the diagonal of L. For image segmentation, Yu and Shi [69] use the modified Laplacian $L + (1/2)D$, which can be indefinite. (Each negative edge contributes $-2w_{ij}x_ix_j$ to the objective function.) For surface reconstruction, Kolluri, Shewchuk, and O'Brien [41] use $L + D$, and find that it offers better partitions and faster eigenvector computation than the standard Laplacian L. Following the example of normalized cuts, both papers choose the vertex masses to be $m_i = \sum_{k \neq i} |w_{ik}|$, so the mass matrix M has the same diagonal as $L + D$.

G.2 Negative Edge Weights and Isoperimetric Partitioning

Negative edge weights can cause the Laplacian matrix to have negative eigenvalues. If that happens, the ellipses in Figure 11 become hyperbolae, and the relaxed optimization problem (24) has no minimum: $x^\mathsf{T}Lx$ approaches $-\infty$ as x approaches ∞v_1, where v_1 is the eigenvector associated with the most negative eigenvalue. Unlike in spectral partitioning, no constraint prevents the components of x from growing arbitrarily large. This is a reminder that relaxing a discrete optimization problem can bring unintended consequences.

We can solve this problem by changing the diagonal of the Laplacian matrix as described in Section G.1. The procedure is slightly more complicated for a 0-1 indicator vector than a signed indicator vector. Observe that in a discrete problem where each x_i is zero or one, for any diagonal matrix D,

$$x^\mathsf{T}Dx = \mathbf{1}^\mathsf{T}Dx.$$

Therefore, minimizing $x^\mathsf{T}Lx$ is equivalent to minimizing

$$x^\mathsf{T}(L + D)x - \mathbf{1}^\mathsf{T}Dx, \tag{61}$$

where we can choose D at our convenience.

The new objective function (61) does not change the minimum of the discrete optimization problem (23), but it can change the minimum of the relaxed problem—or cause it to have a minimum, where before it did not. If D is chosen by Equation (60), then $L + D$ is positive semidefinite, so a minimum exists.

Let's reprise the derivation in Section F.1. For x to minimize the objective function (61) subject to the balance constraint $\mathbf{1}^\mathsf{T}Mx = \mu$ and one constraint for each fixed vertex, a necessary condition is

$$2(L + D)x = D\mathbf{1} + \beta M\mathbf{1} + \sum_{i=j+1}^{n} \gamma_i e_i, \tag{62}$$

where β and each γ_i are Lagrange multipliers. Observe the similarity to the condition (50). Working the extra term $D\mathbf{1}$ through the derivation, we find that we can solve Equation (62) by solving the linear system

$$\mathbb{L}\tilde{x} = b - \frac{D_{jj}}{2M_{jj}} M_{xx}\mathbf{1} + \frac{1}{2}D_{xx}\mathbf{1}$$

for $\tilde{x}$, where $\mathbb{L}$ and b are defined by replacing L with $L + D$ in Expressions (55) and (56), M_{jj} and M_{xx} are as in (51), and D_{jj} and D_{xx} are defined analogously. Then recover x_j from the balance constraint (52).

One nice advantage of negative edge weights is that, as the matrix $L + D$ is positive definite, there is no need to fix a vertex value as the standard isoperimetric partitioning algorithm does.

H Computing Eigenvectors

Computing the Fiedler vector v_2 (and perhaps additional eigenvectors, for purposes described in Sections D and E) is complicated. Unless you are an expert, you are advised not to try to implement it yourself, especially if the Laplacian matrix is sparse and large.

The usual first step is to compute (part or all of) a tridiagonal matrix $T = Q^{\mathsf{T}}LQ$, where Q is orthonormal. For dense or small matrices, this step is done by Householder transformations [27, Sections 5.1 and 8.2.1]; for large sparse matrices, by Lanczos iterations [43]. The matrix T has the same eigenvalues as L, and its eigenvectors are $Q^{\mathsf{T}}v_i$. Next, find the eigenvalue λ_2, typically by bisection on T [27, Section 8.4.1]. Finally, find the eigenvector v_2 by inverse iteration on L [27, Section 7.7.8] or its more quickly convergent cousin, Rayleigh quotient iteration [27, Section 8.4.2] (both of which entail solving a symmetric, but perhaps not positive definite, linear system). If λ_2 is computed with enough accuracy, a few iterations suffice to compute v_2. If a rough approximation of v_2 is available, one iteration often suffices to refine it with good accuracy. One way to obtain a rough approximation is to first compute the eigenvector of T associated with λ_2, then premultiply it by Q. Fast implementations of these operations (with Householder transformations, not Lanczos iterations) are available in LAPACK [4].

Lanczos iterations have a performance advantage for large, sparse matrices because it is not necessary to compute all of T; the first $\Theta(\sqrt{n})$ rows and columns of T comprise a submatrix whose least and greatest eigenvalues are likely to be good estimates of L's least and greatest eigenvalues. Moreover, even an incomplete T can provide a rough estimate of v_2: the extreme eigenvectors of the incomplete T, premultiplied by the incomplete Q, yield estimates of the extreme eigenvectors of L.

The Lanczos algorithm presents two major difficulties. The first is devising a reliable stopping condition for the Lanczos iterations. A vexing problem is *misconvergence*, wherein the least eigenvalue of the growing portion of T computed so far appears to have converged, but is actually lingering temporarily near an eigenvalue of L that is not quite the least. Parlett, Simon, and Stringer [56] provide an invaluable guide to implementing the Lanczos algorithm, including an investigation of stopping conditions and a faster method for computing λ_2 that takes advantage of properties of the Lanczos algorithm.

The second difficulty is the tendency of floating-point roundoff to cause the columns of Q to lose orthogonality. Parlett and Scott [55] discuss a selective orthogonalization algorithm that overcomes this problem, and Hendrickson and Leland [32] discuss how to tailor the algorithm for spectral partitioning.

For very large, sparse matrices, there may not be enough memory to store the $\Theta(\sqrt{n})$ columns of Q necessary for orthogonalization. Barnard and Simon [6] suggest an alternative algorithm that also uses Rayleigh quotient iteration, but instead of the Lanczos process or a tridiagonalization, they use a multilevel method to obtain initial estimates of λ_2 and v_2: they coarsen the graph G with edge contractions, find the Fiedler vector for the contracted graph recursively, and interpolate it onto G. At the coarsest level of the recursion, they use the Lanczos algorithm.

To make it easier for these algorithms to isolate λ_2, replace L with $L' = L + \frac{3}{n}(\max_i L_{ii})\mathbf{1}\mathbf{1}^{\mathsf{T}}$, thereby shifting the eigenvalue of $v_1 = \mathbf{1}$ from zero to something greater than λ_n while leaving the other eigenvalues and all the eigenvectors unchanged. By this means, λ_2 is the smallest eigenvalue of L', making it easier to isolate λ_2 with the Lanczos algorithm and inverse iteration. If L is sparse, it is foolish to form the dense matrix L' explicitly, but it is easy to compute a matrix-vector product $L'z$ without ever forming L', so Lanczos iterations, inverse iterations, and Rayleigh quotient iterations remain efficient.

Both the Lanczos algorithm and the multilevel Rayleigh quotient iteration algorithm are implemented in the partitioner Chaco, whose User's Guide [32] has an excellent discussion of the fine points of both algorithms (and other partitioning algorithms too).

I Faster Bipartite Graph Partitioning via Singular Value Decompositions

Spectral partitioning can work on bipartite graphs just like any other graph. However, Dhillon [12] points out that it is sometimes more efficient to compute a singular value decomposition instead of a standard eigendecomposition, thereby taking advantage of the graph's being bipartite.

Suppose you have a collection W_ℓ of written works, a set W_r of words that appear in those works, and a bipartite graph $G = (W_\ell, W_r, \mathcal{E})$ whose edges $\mathcal{E}$ connect a word to a work if the former appears in the latter. You would like to cluster together works that contain similar words, and simultaneously cluster together words that frequently appear in the same works.

Let A be a $|W_\ell| \times |W_r|$ matrix that expresses the weights of the edges in $\mathcal{E}$. In document clustering, the edge weights typically depend on word frequency. A common choice is $A_{ij} = o_{ij} \log(|W_\ell|/|W_{\ell j}|)$, where o_{ij} is the number of occurrences of word j in work i and $W_{\ell j}$ is the set of works that contain word j.

The Laplacian and mass matrices are

$$L = \begin{bmatrix} D_\ell & -A \\ -A^\mathsf{T} & D_r \end{bmatrix}, \qquad M = \begin{bmatrix} M_\ell & 0 \\ 0 & M_r \end{bmatrix}.$$

D_ℓ and D_r are diagonal matrices whose diagonal components, as usual, are chosen so that the components in each row or column of L sum to zero.

We could compute the eigenvectors of the generalized eigensystem $Lv = \lambda Mv$ by the methods described in Section H, but that is unnecessarily slow, because L is much larger than A. Dhillon observes that in the special case where $M_\ell = D_\ell$ and $M_r = D_r$ (yielding the normalized cut, described in Section A), it is faster to derive the eigenvectors from the singular vectors of the normalized adjacency matrix $D_\ell^{-1/2} A D_r^{-1/2}$.

To extend Dhillon's observation to general positive vertex masses, recall from Section F.3 that changing the diagonal of the Laplacian matrix does not change the solution of the discrete optimization problem. We take advantage of this by choosing $D_\ell = M_\ell$ and $D_r = M_r$. Expanding the system $Lv = \lambda Mv$ gives

$$\begin{aligned} M_\ell v_\ell - A v_r &= \lambda M_\ell v_\ell, \\ M_r v_r - A^\mathsf{T} v_\ell &= \lambda M_r v_r. \end{aligned}$$

Premultiplying these equations by $M_\ell^{-1/2}$ or $M_r^{-1/2}$ and substituting $v_\ell = M_\ell^{-1/2} u_\ell$ and $v_r = M_r^{-1/2} u_r$ gives

$$\begin{aligned} M_\ell^{-1/2} A M_r^{-1/2} u_r &= (1-\lambda) u_\ell, \\ M_r^{-1/2} A^\mathsf{T} M_\ell^{-1/2} u_\ell &= (1-\lambda) u_r. \end{aligned}$$

These identities show that u_ℓ and u_r are, by definition, left and right singular vectors of $M_\ell^{-1/2} A M_r^{-1/2}$, with the singular value $1 - \lambda$.

To find the eigenvector v_i associated with the ith smallest eigenvalue, compute the left and right singular vectors u_ℓ and u_r associated with the ith largest singular value of $M_\ell^{-1/2} A M_r^{-1/2}$ (with LAPACK [4] or the sparse singular value decomposition packages PROPACK [44] or SVDPACK [7]). Then set

$$v_i = \begin{bmatrix} M_\ell^{-1/2} u_\ell \\ M_r^{-1/2} u_r \end{bmatrix}.$$

Any of the methods for partitioning and clustering described in this report then apply.

References

[1] Noga Alon and Vitali D. Milman. λ_1, *Isoperimetric Inequalities for Graphs, and Superconcentrators*. Journal of Combinatorial Theory, Series B **38**(1):73–88, February 1985.

[2] Charles J. Alpert, Andrew B. Kahng, and So-Zen Yao. *Spectral Partitioning with Multiple Eigenvectors*. Discrete Applied Mathematics **90**(1–3):3–26, January 1999.

[3] Christoph Ambühl, Monaldo Mastrolilli, and Ola Svensson. *Inapproximability Results for Sparsest Cut, Optimal Linear Arrangement, and Precedence Constrained Scheduling*. 48th Annual Symposium on Foundations of Computer Science (Providence, Rhode Island), pages 329–337. IEEE Computer Society Press, October 2007.

[4] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania, 1992.

[5] Sanjeev Arora, Satish Rao, and Umesh Vazirani. *Expander Flows, Geometric Embeddings and Graph Partitioning*. Journal of the ACM **56**(2):5, April 2009.

[6] Steven T. Barnard and Horst D. Simon. *A Fast Multilevel Implementation of Recursive Spectral Bisection for Partitioning Unstructured Problems*. Concurrency: Practice and Experience **6**(2):101–117, 1994.

[7] Michael W. Berry. *Large Scale Singular Value Computations*. International Journal of Supercomputer Applications **6**(1):13–49, March 1992.

[8] Thang Nguyen Bui and Curt Jones. *Finding Good Approximate Vertex and Edge Partitions Is NP-Hard*. Information Processing Letters **42**(3):153–159, May 1992.

[9] Jeff Cheeger. *A Lower Bound for the Smallest Eigenvalue of the Laplacian*. Problems in Analysis (Papers dedicated to Salomon Bochner, 1969), pages 195–199. Princeton University Press, Princeton, New Jersey, 1970.

[10] Boris V. Cherkassky, Andrew V. Goldberg, Paul Martin, Joao C. Setubal, and Jorge Stolfi. *Augment or Push: A Computational Study of Bipartite Matching and Unit-Capacity Flow Algorithms*. Journal of Experimental Algorithmics **3**(8), 1998.

[11] André-Louis Cholesky. *Sur la Résolution Numérique des Systèmes d'Équations Linéaires*. Manuscript. Subsequently published in *Bulletin de la Sabix* **39**:81–95, 2005, 1910.

[12] Inderjit Dhillon. *Co-Clustering Documents and Words Using Bipartite Spectral Graph Partitioning*. Proceedings of the Seventh International Conference on Knowledge Discovery and Data Mining (San Francisco, California), pages 269–274, August 2001.

[13] Krzystof Diks, Hristo N. Djidjev, Ondrej Sykora, and Imrich Vrto. *Edge Separators of Planar and Outerplanar Graphs with Applications*. Journal of Algorithms **14**:258–279, 1993.

[14] Irit Dinur. *The PCP Theorem by Gap Amplification*. Journal of the ACM **54**(3):12, June 2007.

[15] Hristo Nicolov Djidjev. *On the Problem of Partitioning Planar Graphs*. SIAM Journal on Algebraic Discrete Methods **3**(2):229–240, 1982.

[16] Jozef Dodziuk. *Difference Equations, Isoperimetric Inequality and Transience of Certain Random Walks*. Transactions of the American Mathematical Society **284**(2):787–794, August 1984.

[17] Wilm E. Donath and Alan J. Hoffman. *Algorithms for Partitioning of Graphs and Computer Logic Based on Eigenvectors of Connection Matrices*. IBM Technical Disclosure Bulletin **15**(3):938–944, August 1972.

[18] _____. *Lower Bounds for the Partitioning of Graphs*. IBM Journal of Research and Development **17**(5):420–425, 1973.

[19] Peter Elias, Amiel Feinstein, and Claude Shannon. *A Note on the Maximum Flow through a Network*. IRE Transactions on Information Theory **2**(4):117–119, December 1956.

[20] Uriel Feige and Robert Krauthgamer. *A Polylogarithmic Approximation of the Minimum Bisection*. SIAM Journal on Computing **31**(4):1090–1118, August 2002.

[21] Charles M. Fiduccia and Robert M. Mattheyses. *A Linear-Time Heuristic for Improving Network Partitions*. Proceedings of the 19th Design Automation Conference (Las Vegas, Nevada), pages 175–181, June 1982.

[22] Miroslav Fiedler. *Algebraic Connectivity of Graphs*. Czechoslovak Mathematical Journal **23**(98):298–305, 1973.

[23] _____. *A Property of Eigenvectors of Nonnegative Symmetric Matrices and its Application to Graph Theory*. Czechoslovak Mathematical Journal **25**(4):619–633, 1975.

[24] Lester R. Ford, Jr. and Delbert R. Fulkerson. *Maximal Flow through a Network*. Canadian Journal of Mathematics **8**:399–404, 1956.

[25] Michael R. Garey, David S. Johnson, and Larry J. Stockmeyer. *Some Simplified NP-Complete Graph Problems*. Theoretical Computer Science **1**(3):237–267, February 1976.

[26] Andrew V. Goldberg and Robert E. Tarjan. *A New Approach to the Maximum Flow Problem*. Journal of the Association for Computing Machinery **35**(4):921–940, October 1988.

[27] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*, second edition. The Johns Hopkins University Press, 1989.

[28] Gene H. Golub and Urs von Matt. *Quadratically Constrained Least Squares and Quadratic Problems*. Numerische Mathematik **59**(1):561–580, December 1991.

[29] Leo Grady and Eric L. Schwartz. *Isoperimetric Partitioning: A New Algorithm for Graph Partitioning*. SIAM Journal on Scientific Computing **27**(6):1844–1866, 2006.

[30] Stephen Guattery and Gary L. Miller. *On the Quality of Spectral Separators*. SIAM Journal on Matrix Analysis and Applications **19**(3):701–719, July 1998.

[31] Kenneth M. Hall. *An r-dimensional Quadratic Placement Algorithm*. Management Science **17**(3):219–229, November 1970.

[32] Bruce Hendrickson and Robert Leland. *The Chaco User's Guide, Version 2.0*. Technical Report SAND95-2344, Sandia National Laboratories, Albuquerque, New Mexico, July 1995.

[33] ———. *An Improved Spectral Graph Partitioning Algorithm for Mapping Parallel Computations*. SIAM Journal on Scientific Computing **16**(2):452–469, 1995.

[34] Bruce Hendrickson, Robert Leland, and Rafael Van Driessche. *Skewed Graph Partitioning*. Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing (Minneapolis, Minnesota). SIAM Journal on Scientific Computing, March 1997.

[35] Magnus R. Hestenes and Eduard Stiefel. *Methods of Conjugate Gradients for Solving Linear Systems*. Journal of Research of the National Bureau of Standards **49**:409–436, 1952.

[36] John E. Hopcroft and Richard M. Karp. *An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs*. SIAM Journal on Computing **2**(4):225–231, 1973.

[37] Richard M. Karp. *Reducibility among Combinatorial Problems*. Complexity of Computer Computations (Raymond E. Miller and James W. Thatcher, editors), pages 85–103. Plenum Press, New York, 1972.

[38] Brian W. Kernighan and Shen Lin. *An Efficient Heuristic Procedure for Partitioning Graphs*. Bell System Technical Journal **49**(2):291–307, February 1970.

[39] Dénes Kőnig. *Gráfok és Mátrixok*. Matematikai és Fizikai Lapok **38**:116–119, 1931.

[40] Subhash Khot. *Ruling Out PTAS for Graph Min-Bisection, Dense k-Subgraph, and Bipartite Clique*. SIAM Journal on Computing **36**(4):1025–1071, 2006.

[41] Ravikrishna Kolluri, Jonathan Richard Shewchuk, and James F. O'Brien. *Spectral Surface Reconstruction from Noisy Point Clouds*. Symposium on Geometry Processing 2004 (Nice, France), pages 11–21. Eurographics Association, July 2004.

[42] Tsz Chiu Kwok, Lap Chi Lau, Yin Tat Lee, Shayan Oveis Gharan, and Luca Trevisan. *Improved Cheeger's Inequality: Analysis of Spectral Partitioning Algorithms through Higher Order Spectral Gap*. Proceedings of the Forty-Fifth Annual Symposium on Theory of Computing (Palo Alto, California), pages 11–20, June 2013.

[43] Cornelius Lanczos. *An Iteration Method for the Solution of the Eigenvalue Problem of Linear Differential and Integral Operators*. Journal of Research of the National Bureau of Standards **45**:255–282, 1950.

[44] Rasmus Munk Larsen. *Lanczos Bidiagonalization with Partial Reorthogonalization*. Technical Report DAIMI PB-357, Department of Computer Science, University of Århus, Århus, Denmark, October 1998.

[45] Gregory F. Lawler and Alan D. Sokal. *Bounds on the L^2 Spectrum for Markov Chains and Markov Processes: A Generalization of Cheeger's Inequality*. Transactions of the American Mathematical Society **309**(2):557–580, October 1988.

[46] James R. Lee, Shayan Oveis Gharan, and Luca Trevisan. *Multi-Way Spectral Partitioning and Higher-Order Cheeger Inequalities*. Proceedings of the Forty-Fourth Annual Symposium on Theory of Computing (New York, New York), pages 1117–1130, May 2012.

[47] Richard J. Lipton, Donald J. Rose, and Robert Endre Tarjan. *Generalized Nested Dissection*. SIAM Journal on Numerical Analysis **16**(2):346–358, April 1979.

[48] Richard J. Lipton and Robert Endre Tarjan. *A Separator Theorem for Planar Graphs*. SIAM Journal on Applied Mathematics **36**(2):177–189, 1979.

[49] Stuart P. Lloyd. *Least Squares Quantization in PCM*. IEEE Transactions on Information Theory **28**(2):129–137, March 1982.

[50] David W. Matula and Farhad Shahrokhi. *Sparsest Cuts and Bottlenecks in Graphs*. Discrete Applied Mathematics **27**(1–2):113–123, May 1990.

[51] Gary L. Miller, Shang-Hua Teng, William Thurston, and Steven A. Vavasis. *Automatic Mesh Partitioning*. Graph Theory and Sparse Matrix Computation (Alan George, John R. Gilbert, and Joseph W. H. Liu, editors), pages 57–84. Springer-Verlag, New York, 1993.

[52] Bojan Mohar. *Isoperimetric Numbers of Graphs*. Journal of Combinatorial Theory, Series B **47**(3):274–291, December 1989.

[53] Bojan Mohar and Svatopluk Poljak. *Eigenvalues and the Max-Cut Problem*. Czechoslovak Mathematical Journal **40**(2):343–352, 1990.

[54] Andrew Y. Ng, Michael I. Jordan, and Yair Weiss. *On Spectral Clustering: Analysis and an Algorithm*. Advances in Neural Information Processing Systems 14 (Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani, editors), pages 849–856. The MIT Press, September 2002.

[55] Beresford N. Parlett and David Scott. *The Lanczos Algorithm with Selective Orthogonalization*. Mathematics of Computation **33**:217–238, 1979.

[56] Beresford N. Parlett, Horst D. Simon, and L. M. Stringer. *On Estimating the Largest Eigenvalue With the Lanczos Algorithm*. Mathematics of Computation **38**(157):153–165, January 1982.

[57] Alex Pothen. *An Analysis of Spectral Graph Partitioning via Quadratic Assignment Problems*. Domain Decomposition Methods in Scientific and Engineering Computing: Proceedings of the Seventh International Conference on Domain Decomposition (David E. Keyes and Jinchao Xu, editors), Contemporary Mathematics, volume 180, pages 105–110. American Mathematical Society, Providence, Rhode Island, October 1993.

[58] Alex Pothen, Horst D. Simon, and Kang-Pu Liou. *Partitioning Sparse Matrices with Eigenvectors of Graphs*. SIAM Journal on Matrix Analysis and Applications **11**(3):430–452, July 1990.

[59] Omer Reingold. *Undirected Connectivity in Log-Space*. Journal of the ACM **55**(4):17, September 2008.

[60] Franz Rendl and Henry Wolkowicz. *A Projection Technique for Partitioning the Nodes of a Graph*. Annals of Operations Research **58**(3):155–179, May 1995.

[61] Jianbo Shi and Jitendra Malik. *Normalized Cuts and Image Segmentation*. IEEE Transactions on Pattern Analysis and Machine Intelligence **22**(8):888–905, 2000.

[62] Horst D. Simon. *Partitioning of Unstructured Problems for Parallel Processing*. Computing Systems in Engineering **2**(2/3):135–148, 1991.

[63] Alistair Sinclair and Mark Jerrum. *Approximate Counting, Uniform Generation and Rapidly Mixing Markov Chains*. Information and Computation **82**(1):93–133, July 1989.

[64] Daniel A. Spielman and Shang-Hua Teng. *Spectral Partitioning Works: Planar Graphs and Finite Element Meshes*. 37th Annual Symposium on Foundations of Computer Science (Burlington, Vermont), pages 96–105. IEEE Computer Society Press, October 1996.

[65] ———. *Spectral Partitioning Works: Planar Graphs and Finite Element Meshes*. Technical Report CSD-96-898, Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley, California, November 1996.

[66] Mechthild Stoer and Frank Wagner. *A Simple Min-Cut Algorithm*. Journal of the ACM **44**(4):585–591, July 1997.

[67] Takeshi Tokuyama and Jun Nakano. *Geometric Algorithms for the Minimum Cost Assignment Problem*. Random Structures and Algorithms **6**(4):393–406, July 1995.

[68] Rafael Van Driessche and Dirk Roose. *An Improved Spectral Bisection Algorithm and its Application to Dynamic Load Balancing*. Parallel Computing **21**(1):29–48, January 1995.

[69] Stella X. Yu and Jianbo Shi. *Understanding Popout through Repulsion*. IEEE Conference on Computer Vision and Pattern Recognition (Kauai, Hawaii), volume 2, pages 752–757, December 2001.

[70] ———. *Multiclass Spectral Clustering*. Ninth IEEE International Conference on Computer Vision (Nice, France), volume 2, pages 313–319, October 2003.