22 Spectral Graph Clustering

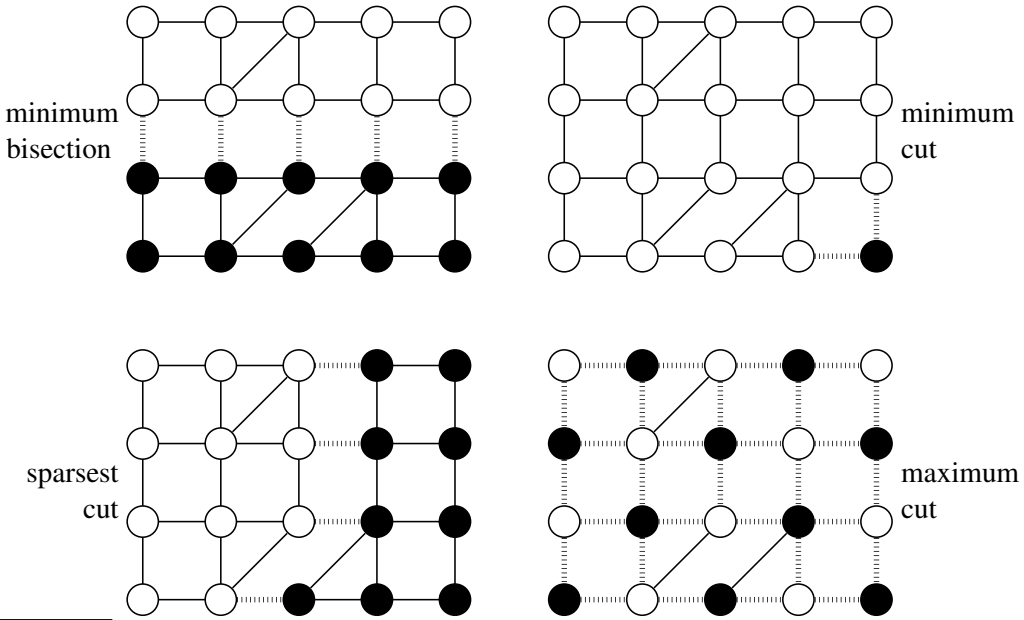
SPECTRAL GRAPH CLUSTERING

Input: Weighted, undirected graph $G = (V, E)$. No self-edges.
w_{ij} = weight of edge $(i, j) = (j, i)$; zero if $(i, j) \notin E$.

[Think of the edge weights as a similarity measure. A big weight means that the two vertices want to be in the same cluster. So the circumstances are the opposite of the last lecture on clustering. Then, we had a distance or dissimilarity function, so small numbers meant that points wanted to stay together. Today, big numbers mean that vertices want to stay together.]

Goal: Cut G into 2 (or more) pieces G_i of similar sizes,
but don't cut too much edge weight.
[That's a vague goal. There are many ways to make this precise.
Here's a typical goal, which we'll solve approximately.]
e.g., Minimize the sparsity $\frac{\text{Cut}(G_1,G_2)}{\text{Mass}(G_1)\,\text{Mass}(G_2)}$, aka cut ratio
where $\text{Cut}(G_1, G_2)$ = total weight of cut edges
$\text{Mass}(G_1)$ = # of vertices in G_1 OR assign masses to vertices

[The denominator "$\text{Mass}(G_1)\,\text{Mass}(G_2)$" penalizes imbalanced cuts.]



graph.pdf [Four cuts. All edges have weight 1.
Upper left: the minimum bisection; a bisection is perfectly balanced.
Upper right: the minimum cut. Usually very unbalanced; not what we want.
Lower left: the sparsest cut, which is good for many applications.
Lower right: the maximum cut; in this case also the maximum bisection.]

Sparsest cut, min bisection, max cut all NP-hard.
[Today we will look for an approximate solution to the sparsest cut problem.]

[We will turn this combinatorial graph cutting problem into algebra.]

Let $n = |V|$. Let $y \in \mathbb{R}^n$ be an indicator vector:

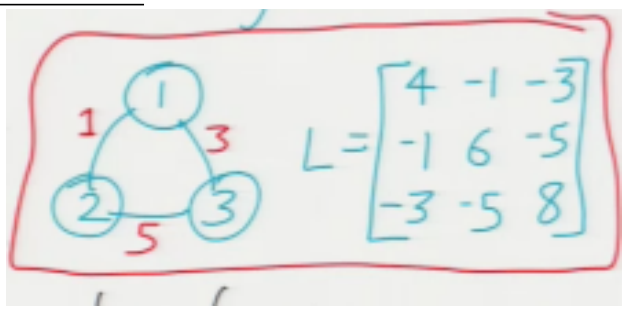
$$y_i = \begin{cases} 1 & \text{vertex } i \in G_1, \\ -1 & \text{vertex } i \in G_2. \end{cases}$$

Then $w_{ij}\dfrac{(y_i - y_j)^2}{4} = \begin{cases} w_{ij} & (i,j) \text{ is cut}, \\ 0 & (i,j) \text{ is not cut}. \end{cases}$

$$\begin{aligned} \text{Cut}(G_1, G_2) &= \sum_{(i,j)\in E} w_{ij}\frac{(y_i - y_j)^2}{4} && \text{[This is quadratic, so let's try to write it with a matrix.]} \\ &= \frac{1}{4}\sum_{(i,j)\in E}\left(w_{ij}\,y_i^2 - 2w_{ij}\,y_i\,y_j + w_{ij}\,y_j^2\right) \\ &= \frac{1}{4}\left(\underbrace{\sum_{(i,j)\in E} -2w_{ij}\,y_i\,y_j}_{\text{off-diagonal terms}} + \underbrace{\sum_{i=1}^{n} y_i^2 \sum_{k\neq i} w_{ik}}_{\text{diagonal terms}}\right) \\ &= \frac{y^\top L y}{4}, \end{aligned}$$

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

L is symmetric, $n \times n$ Laplacian matrix for G.



[Draw this by hand graphexample.png]

[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.]
[We see that minimizing the weight of the cut is equivalent to minimizing the Laplacian quadratic form $y^\top L y$. This lets us turn graph partitioning into a problem in matrix algebra.]
[Usually we assume there are no negative weights, in which case $\text{Cut}(G_1, G_2)$ can never be negative, so it follows that L is positive semidefinite.]

Define $\mathbf{1} = [1 \quad 1 \quad \ldots \quad 1]^\top$; then $L\mathbf{1} = \mathbf{0}$, so [It's easy to check that each row of L sums to zero.]
$\mathbf{1}$ is an eigenvector of L with eigenvalue 0.

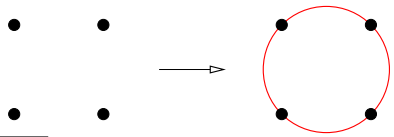
[If G is a connected graph and all the edge weights are positive, then this is the only zero eigenvalue. But if G is not connected, L has one zero eigenvalue for each connected component of G. It's easy to prove, but time prevents me.]

Bisection: exactly $n/2$ vertices in G_1, $n/2$ in G_2. Write $\mathbf{1}^\top y = 0$.
[So we have reduced graph bisection to this constrained optimization problem.]
Minimum bisection:

Find y that minimizes $y^\top L y$		
subject to	$\forall i$, $y_i = 1$ or $y_i = -1$	← binary constraint
and	$\mathbf{1}^\top y = 0$	← balance constraint

Also NP-hard. We relax the binary constraint. → fractional vertices!

[A very common approach in combinatorial optimization algorithms is to relax some of the constraints so a discrete problem becomes a continuous problem. Intuitively, this means that you can put $1/3$ of vertex 7 in graph G_1 and the other $2/3$ of vertex 7 in graph G_2. You can even put $-1/2$ of vertex 7 in graph G_1 and $3/2$ of vertex 7 in graph G_2. This sounds crazy, but the continuous problem is much easier to solve than the combinatorial problem. After we solve it, we will round the vertex values to $+1/-1$, and we'll hope that our solution is still close to optimal.]
[But we can't just drop the binary constraint. We still need *some* constraint to rule out the solution $y = 0$.]
New constraint: y must lie on hypersphere of radius $\sqrt{n}$.

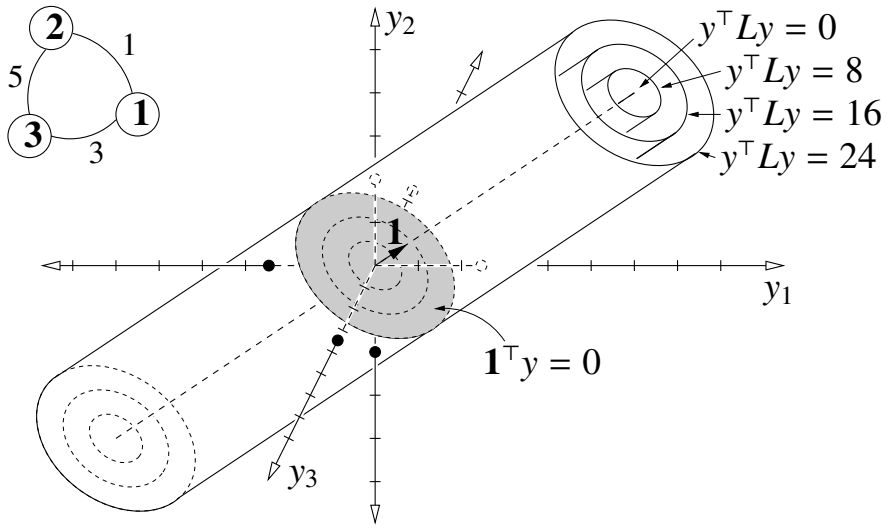


[Draw this by hand. circle.pdf] [Instead of constraining y to lie at a vertex of the hypercube, we constrain y to lie on the hypersphere through those vertices.]

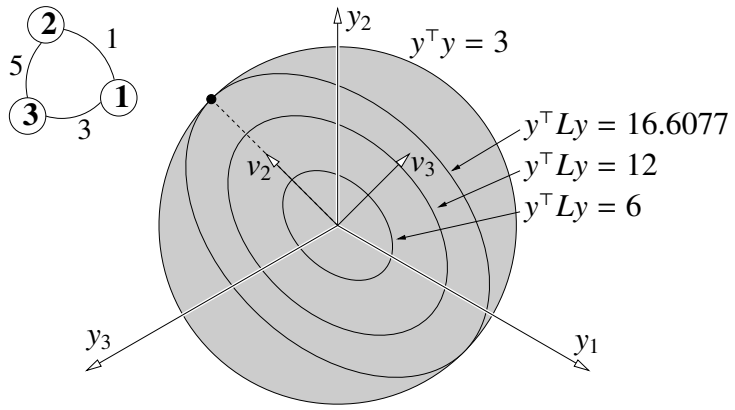
Relaxed problem:

Minimize $y^\top L y$		
subject to	$y^\top y = n$	
and	$\mathbf{1}^\top y = 0$	

$\Big\}$ = Minimize $\dfrac{y^\top L y}{y^\top y}$ = Rayleigh quotient of L & y



cylinder.pdf [The isosurfaces of $y^\top L y$ are elliptical cylinders. The gray cross-section is the hyperplane $\mathbf{1}^\top y = 0$. We seek the point that minimizes $y^\top L y$, subject to the constraints that it lies on the gray cross-section and that it lies on a sphere centered at the origin.]

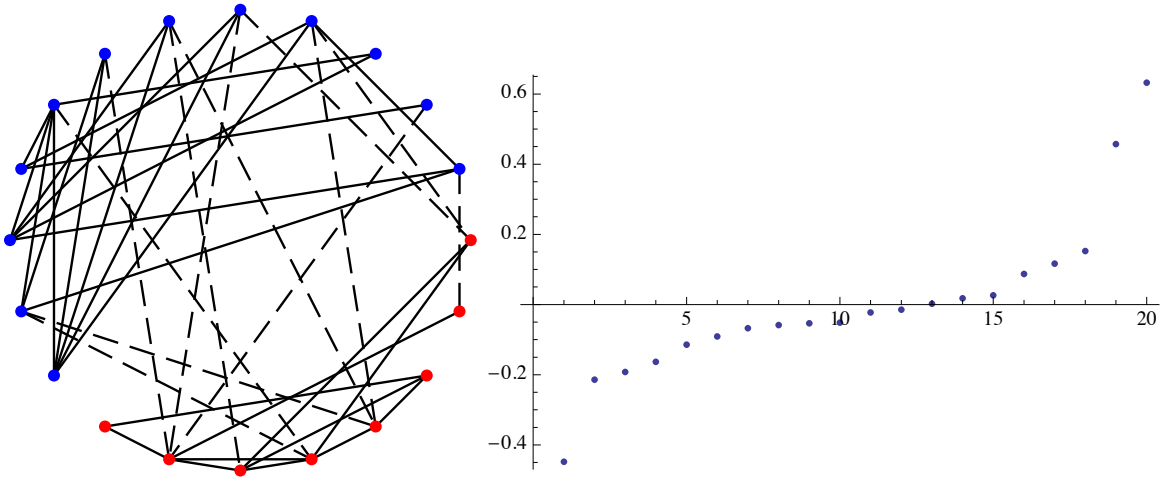


endview.pdf [The same isosurfaces restricted to the hyperplane $\mathbf{1}^\top y = 0$. The solution is constrained to lie on the outer circle.]

[You should remember this Rayleigh quotient from the lecture on PCA. As I said then, when you see a Rayleigh quotient, you should smell eigenvectors nearby. The y that minimizes this Rayleigh quotient is the eigenvector with the smallest eigenvalue. We already know what that eigenvector is: it's $\mathbf{1}$. But that violates our balance constraint. As you should recall from PCA, when you've used the most extreme eigenvector and you need an orthogonal one, the next-best optimizer of the Rayleigh quotient is the next eigenvector.]

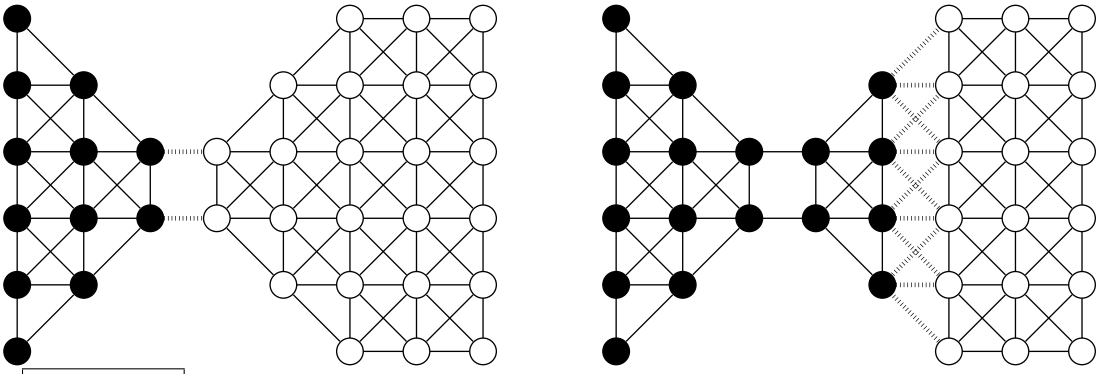
Let λ_2 = second-smallest eigenvalue of L.
Eigenvector v_2 is the Fiedler vector.
[It would be wonderful if every component of the Fiedler vector was 1 or -1, but that happens more or less never. So we round v_2. The simplest way is to round all positive entries to 1 and all negative entries to -1. But in both theory and practice, it's better to choose the threshold as follows.]

Spectral partitioning alg:
- Compute Fiedler vector v_2 of L
- Round v_2 with a sweep cut:
  - = Sort components of v_2.
  - = Try the $n-1$ cuts between successive components. Choose min-sparsity cut.
    [If we're clever about updating the sparsity, we can try all these cuts in time linear in the number of edges in G.]



specgraph.pdf, specvector.pdf [Left: example of a graph partitioned by the sweep cut. Right: what the un-rounded Fiedler vector looks like.]

[One consequence of relaxing the binary constraint is that the balance constraint no longer forces an exact bisection. But that's okay; we're cool with a slightly off-balance constraint if it means we cut fewer edges. Even though our discrete problem was the minimum bisection problem, our relaxed, continuous problem will be an approximation of the sparsest cut problem. This is a bit counterintuitive.]

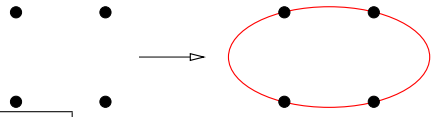


lopsided.pdf [A graph for which an off-balance cut (left) is sparser than a balanced one (right).]

Vertex Masses

[Sometimes you want the notion of balance to accord more prominence to some vertices than others. We can assign masses to vertices.]

Let M be diagonal matrix with vertex masses on diagonal.
New balance constraint: $\mathbf{1}^\top My = 0$.
[This new balance constraint says that G_1 and G_2 should each have the same total mass. It turns out that this new balance constraint is easier to satisfy if we also revise the sphere constraint a little bit.]
New ellipsoid constraint: $y^\top My = \text{Mass}(G) = \sum M_{ii}$.
[Instead of a sphere, now we constrain y to lie on an axis-aligned ellipsoid.]



[Draw this by hand. ellipse.pdf] [The constraint ellipsoid passes through the points of the hypercube.]

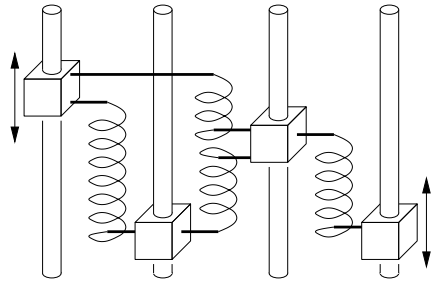
Now solution is Fiedler vector of generalized eigensystem $Lv = \lambda Mv$.
[Most algorithms for computing eigenvectors and eigenvalues of symmetric matrices can easily be adapted to compute eigenvectors and eigenvalues of symmetric *generalized* eigensystems too.]

[For the grad students, here's the most important theorem in spectral graph partitioning.]

Fact: Sweep cut finds a cut w/sparsity $\leq \sqrt{2\lambda_2 \max_i \frac{L_{ii}}{M_{ii}}}$: Cheeger's inequality.
The optimal cut has sparsity $\geq \lambda_2/2$.

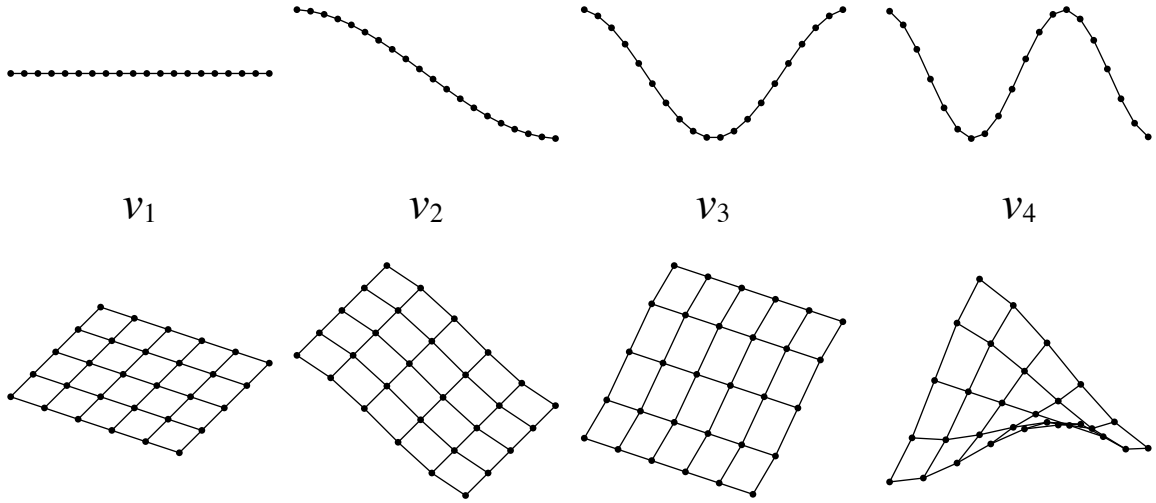
[So the spectral partitioning algorithm is an approximation algorithm, albeit not one with a constant factor of approximation. Cheeger's inequality is a very famous result in spectral graph theory, because it's one of the most important cases where you can relax a combinatorial optimization problem to a continuous optimization problem, round the solution, and still have a provably decent solution to the original combinatorial problem.]

Vibration Analogy



vibrate.pdf

[For intuition about spectral partitioning, think of the eigenvectors as vibrational modes in a physical system of springs and masses. Each vertex models a point mass 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.]



grids.pdf [Vibrational modes in a path graph and a grid graph.]

[These illustrations show the first four eigenvectors for two simple graphs. On the left, we see that the first eigenvector is the eigenvector of all 1's, which represents a vertical translation of all the masses in unison. That's not really a vibration, which is why the eigenvalue is zero. The second eigenvector is the Fiedler vector, which represents the vibrational mode with the lowest frequency. Each component 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. So it makes sense to cut between the positive components and the negative components. The third eigenvector also gives us a nice bisection of the grid graph, entirely different from the Fiedler vector. Some more sophisticated graph clustering algorithms use multiple eigenvectors.]

[I want to emphasize that spectral partitioning takes a global view of a graph. It looks at the whole gestalt of the graph and finds a good cut. By comparison, the clustering algorithms we saw last lecture were much more local in nature, so they're easier to fool.]

Greedy Divisive Clustering

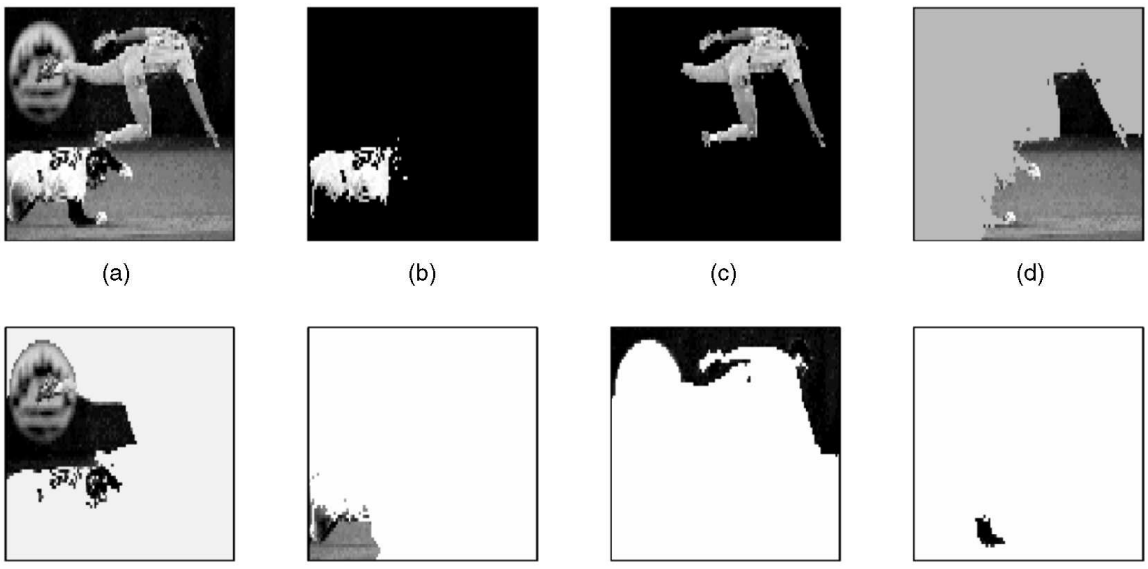
Partition G into 2 subgraphs; recursively cluster them.
[The sparsity is a good criterion for graph clustering. Use G's sparsest cut to divide it into two subgraphs, then recursively cut them. You can stop when you have the right number of clusters, or you could keep going until each subgraph is a single vertex and create a dendrogram.]
Can form a dendrogram, but it may have inversions.
[There's no reason to expect that the sparsity of a subgraph is smaller than the sparsity of the parent graph, so the dendrogram can have inversions. But the hierarchy is still useful for getting an arbitrary number of clusters on demand.]

The Normalized Cut

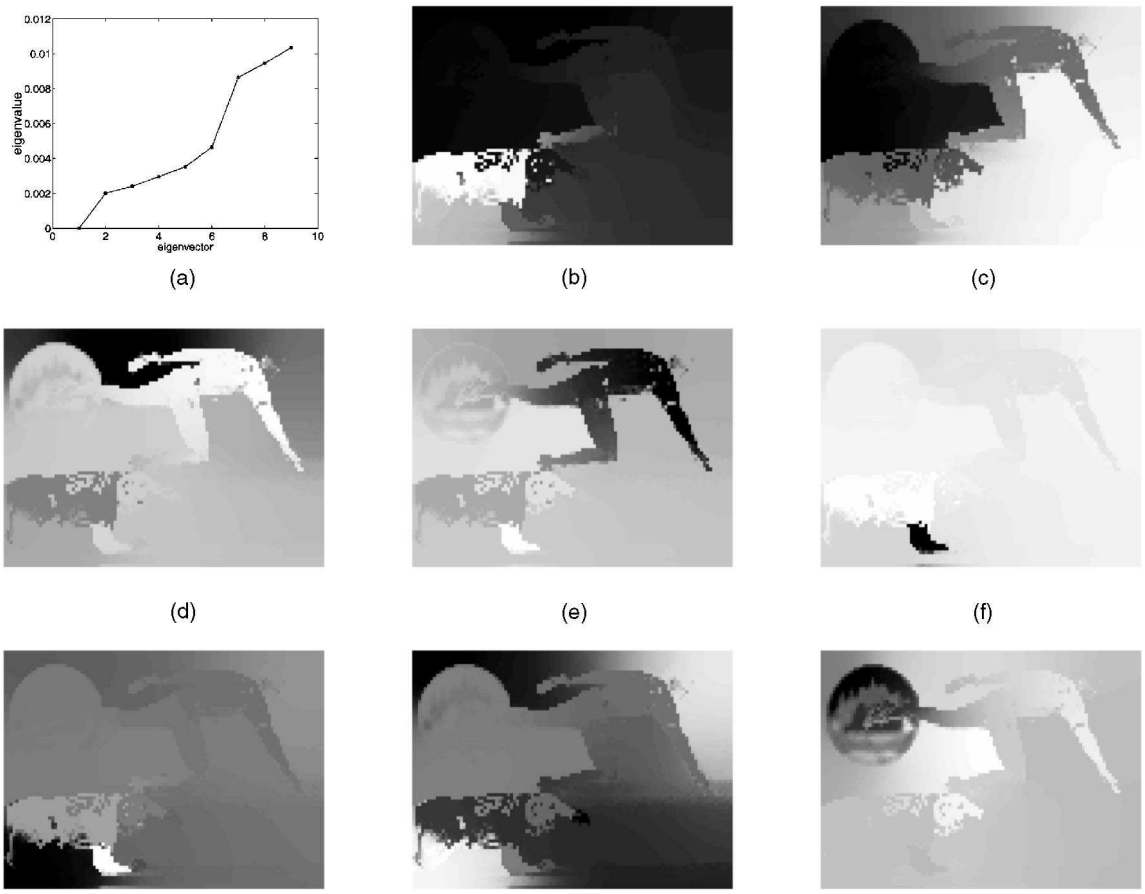
Set vertex i's mass $M_{ii} = L_{ii}$. [Sum of edge weights adjoining vertex i.]
[That is how we define a normalized cut, which turns out to be a good choice for many different applications.]
Popular for image segmentation.
[Image segmentation is the problem of looking at a photograph and separating it into different objects. To do that, we define a graph on the pixels.]
For pixels with coordinate p_i, brightness b_i, use graph weights

$$w_{ij} = \exp\left(-\frac{\|p_i - p_j\|^2}{\alpha} - \frac{|b_i - b_j|^2}{\beta}\right) \qquad \text{or zero if } \|p_i - p_j\| \text{ large.}$$

[We choose a distance threshold, typically less than 4 to 10 pixels apart. Pixels that are far from each other aren't connected. α and β are empirically chosen constants. It often makes sense to choose β proportional to the variance of the brightness values.]



baseballsegment.pdf (Shi and Malik, "Normalized Cut and Image Segmentation")
[A segmentation of a photo of a scene from a baseball game (upper left). The other figures show segments of the image extracted by recursive spectral partitioning.]



baseballvectors.pdf (Shi and Malik) [Eigenvectors 2–9 from the baseball image.]

Invented by [our own] Prof. Jitendra Malik and his student Jianbo Shi.