Spectral Partitioning

Lecturer: Daniel A. Spielman September 26, 2006

6.1 Cut Problems

Graph partitioning problems typically reqire one to divide the vertices of a graph into two or more large pieces, while cutting as few edges as possible. If the graph is weighted, this probably means minimizing the sum of the weights of the cut edges. There are many possible constraints that could be put on the sizes of the pieces produced. For example, the graph bisection problem is to find a set $S \subset V$ such that $|S| = \lfloor |V|/2 \rfloor$ minimizing

$$|\{(i,j) \in E : i \in S, j \notin S\}|.$$

We will refer to a set $S \subseteq V$ as a *cut*, by which we mean the partition of V into S and its complement.

But, a strict bisection is often not exactly what one wants. There are various ways of loosening the bisection constraint. We will consider two.

To define these, let $w_{i,j}$ denote the weight of the edge from i to j. For a set of edges $F \subseteq E$, let $w(F)$ be the sum of the weights of edges in F.

The first is determined by the *sparsity* of a cut, which we define to be

$$\mathrm{sp}(S) \stackrel{\mathrm{def}}{=} \frac{w(\partial(S))}{|S|\,|V-S|},$$

where $\partial(S)$ is the boundary of S: all edges with one endpoint in S and the other outside of S.

This measure weights a cut by the number of vertices inside, and values higher cuts that are more balanced. The *sparsest cut problem* is to find the set S minimizing $\mathrm{sp}(S)$.

A slightly more useful term is given by *conductance*. Following Sinclair and Jerrum, we define conductance by, setting

$$d_i = \sum_j w_{i,j}$$

to be the weighted degree of vertex i, and

$$d(S) \stackrel{\mathrm{def}}{=} \sum_{i \in S} d_i.$$

Then, the conductance of a set S is given by

$$\phi(S) \stackrel{\mathrm{def}}{=} \frac{w(\partial(S))}{\min(d(S), d(V-S))}.$$

We can then consider the problem of finding the set S of minimum conductance.

Unforunately, we cannot solve any of these problems. Not only are they all NP-complete, but no efficient approximation algorithms for them are known. That said, there are many good heuristics. We will see one in the next section.

6.2 Relation to Laplacians

These graph partitioning problems are closely related to problems on Laplacians and normalized Laplacians. To see why, we'll start by examining the sparsest cut problem. We will see how to find approximate solutions to this problem by using one of the greatest ideas in optimization: relaxation.

The idea of relaxation is to take a problem that is fundamentally discrete, and turn it into a continuous problem. Here, our problem requires us to find a set S. To turn the problem into a continuous one, we need to formulate it in an arithmetic fashion. To begin, we consider vectors instead of sets. In particular, we will represent S by the vector χ_S.

For simplicity, let's just consider the unweighted case. In this case, we need a formula for

$$\left|\{(i,j) \in E : i \in S, j \notin S\}\right|.$$

We will use

$$\sum_{(i,j)\in E} (v(i) - v(j))^2.$$

Similarly, for $|S|\,|V - S|$, we will use

$$\sum_{i<j} (v(i) - v(j))^2.$$

So, our problem becomes

$$\min_{v\in\{0,1\}^n} \frac{\sum_{(i,j)\in E}(v(i) - v(j))^2.}{\sum_{i<j}(v(i) - v(j))^2}.$$

To relax this problem, we will allow v to range over all of $\mathbb{R}^n$. This will give a solution of lower value, but it probably won't be 0/1 valued. When we allow v to be real-valued, we see that some simplification is possible: if we added a constant to each entry of v, then the ratio wouldn't change. So, we will normalize by assuming $\sum_i v(i) = 0$. In this case, we also have that

$$\sum_{i<j} (v(i) - v(j))^2 = n \sum v(i)^2.$$

So, our relaxation becomes

$$\begin{aligned}
\min_{v\in\mathbb{R}^n} \frac{\sum_{(i,j)\in E}(v(i) - v(j))^2}{\sum_{i<j}(v(i) - v(j))^2} &= \min_{v\in\mathbb{R}^n, v\perp\mathbf{1}} \frac{\sum_{(i,j)\in E}(v(i) - v(j))^2}{\sum_{i<j}(v(i) - v(j))^2} \\
&= \min_{v\in\mathbb{R}^n, v\perp\mathbf{1}} \frac{\sum_{(i,j)\in E}(v(i) - v(j))^2}{n\sum_i (v(i))^2} \\
&= \min_{v\in\mathbb{R}^n, v\perp\mathbf{1}} \frac{1}{n}\frac{v^T L v}{v^T v}.
\end{aligned}$$

But, by the Courant-Fischer theorem which we learned last lecture, this is just the second-smallest eigenvalue of the Laplacian, divided by n. And, we can compute this. So, we learn that

$$\min_S \text{sp}(S) \geq \lambda_2(L)/n.$$

Moreover, we might hope that by computing the eigenvector of the second-smallest eigenvalue, we might be able to turn it into a set S of low sparsity.

By similar reasoning, it is possible to show that

$$\min_S \phi(S) \geq \lambda_2(\mathcal{L}). \tag{6.1}$$

Moreover, for the normalized Laplacian, there is a clean relationship going in the other direction, knows as Cheeger's Inequality. It was first proved by Cheeger in 1970 for manifolds. The version here for graphs was proved by Jerrum and Sinclair.

Theorem 6.2.1.

$$\min_S \phi(S) \leq \sqrt{2\lambda_2(\mathcal{L})}. \tag{6.2}$$

Not only does the second-smallest eigenvalue of the normalized Laplacian provide an upper bound on $\min_S \phi(S)$, the second-smallest eigenvector can also be used to find a cut.

6.3 Finding a cut

Let v be the eigenvector of the second-smallest eigenvalue of $\mathcal{L}$. Recall that $w = D^{-1/2}v$ is a right-eigenvector of the second-largest eigenvalue of the walk matrix $M = D^{-1}A$.

To find a cut, we first assume that the vertices are ordered so that $v(1) \geq v(2) \geq \cdots \geq v(n)$ (We probably have to permute the vertices to achieve this). We then define the sets

$$S_j = \{1, \ldots, j\}.$$

From the proof of Cheeger's inequality, one can show that for some j,

$$\phi(S_j) \leq \sqrt{2\lambda_2(\mathcal{L})}. \tag{6.3}$$

So, while the cut we find is not necessarily optimal, it does satisfy some guarantee. In particular, (6.1) tells us that if a graph has a set of small conductance, then $\lambda_2(\mathcal{L})$ is small. Conversely, (6.3) tells us that if $\lambda_2(\mathcal{L})$ is small, then the corresponding eigenvector reveals a cut of small, but not necessarily minimum, conductance.

6.4 Disclaimer

At this point, I should mention that there are many other heuristics for finding cuts of low conductance, and there are even algorithms that are known to provide better approximations.

But, I will continute to talk about the spectral approach because:

- it does work unusually well,
- it is easy to compute v_2 if you know a little numerical analysis, and
- I don't have all day.

6.5 Image Segmentation

Everything that I've told you so far has been known since the 80's. Around 2000, Shi and Malik rediscovered some of this material, and applied it to the segmentation of images. This introduced the technique to a new community, and generated quite a bit of interest.

Shi and Malik introduced what they called the Normalized Cut, given by

$$\min_S = \frac{\sum_{(i,j)\in E: i\in S, j\notin S} w_{i,j}}{d(S)d(V-S)},$$

and observed that a relaxation of the normalized cut was given by the normalized Laplacian.

Given an image, they considered the problem of dividing it up into regions. They did this by creating a grid graph with one vertex for each pixel in the image. They then put weights on edges of the graph whose strength was inversely related to their similarity. For example, if each pixel $p(i,j)$ is an element of $\mathbb{R}^3$, say a RGB value, then they would set the weight of the edge from (i,j) to $(i+1,j)$ to

$$e^{-\frac{(p(i,j)-p(i+1,j))^2}{\sigma^2}},$$

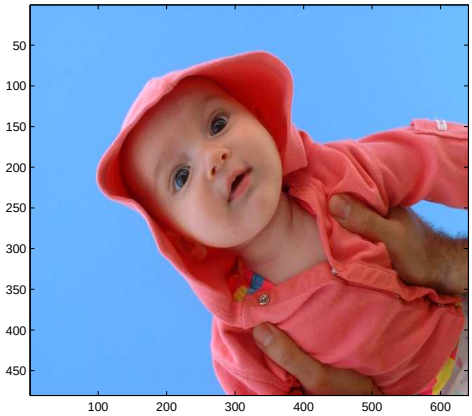
where σ is kept constant throughout the construction. So, the more different the pixels were, the lower the weight of the edge between them would be.

They then obtained eigenvectors corresponding to the top eigenvalues of the walk matrix, and used them to partition the graph. They obtained relatively good results with relatively little work.

Here is an example that I did with picture of my daugher. All the files should be listed on the class web page.

I'll cut out three-fourths of the pictures so that the size of the graph I get is more managable.

```
>> img = imread('airplane.JPG');
>> size(img)

ans =

         960        1280

>> img = img(1:2:end, 1:2:end,:)
>> size(img)

ans =

   480   640     3
```




```
>> image(img)
```

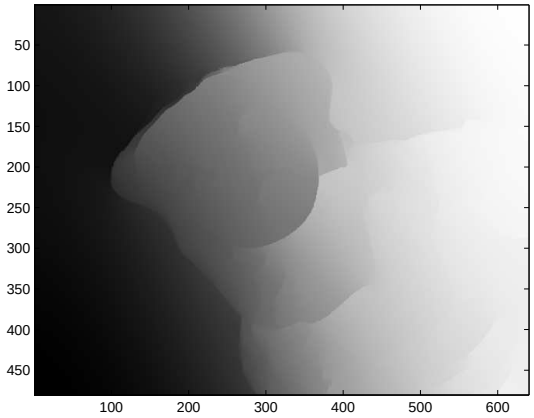
The code `imgeig` returns the adjacency matrix, a, and the first k eigenvectors. The parameter *support* determines how far apart pixels can be and still be connected by an edge. As I explained that only adjacent pixels are connected, we should use support 1. In class, I actually used support 2.

```
>> [V,E,a] = imgeig(img, 6, 1);

xsz =

   480


ysz =

   640

Iteration 1: a few Ritz values of the 20-by-20 matrix:
     0
     0
     0
     0
     0
     0
     0

Iteration 2: a few Ritz values of the 20-by-20 matrix:
   1.0e+08 *

    0.0004
    0.0005
```

```
    0.0008
    0.0011
    0.0020
    0.0025
    1.0000
```

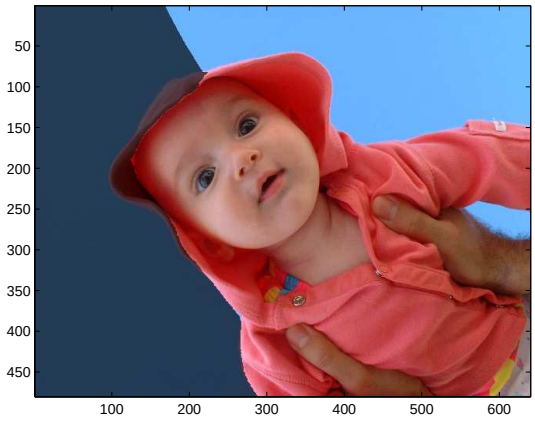
Now, let's look at the second eigenvector in greyscale.

```
>> w = V(:,2);
>> w = w-min(w);
>> w = w / max(w);
>> w = reshape(w,480,640);
>> wimg(:,:,1) = w;
>> wimg(:,:,2) = w;
>> wimg(:,:,3) = w;
>> image(wimg)
```



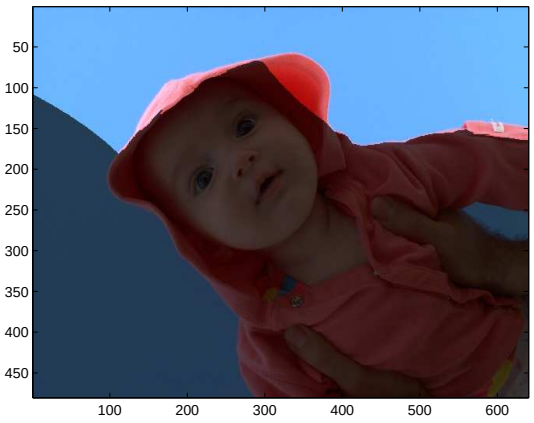
And, here is the cut generated by this eigenvector

```
>> w = w(:);
>> size(w)

ans =

       307200            1

>> S = sparsecut(a,w);
>> imcut(img,S)
```

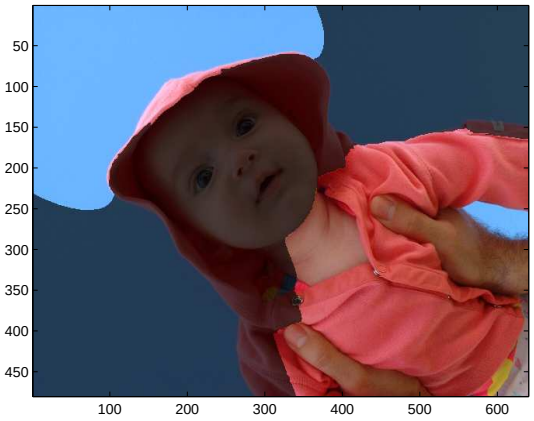


And, here are the cuts generated by the third through fifth eigenvectors.

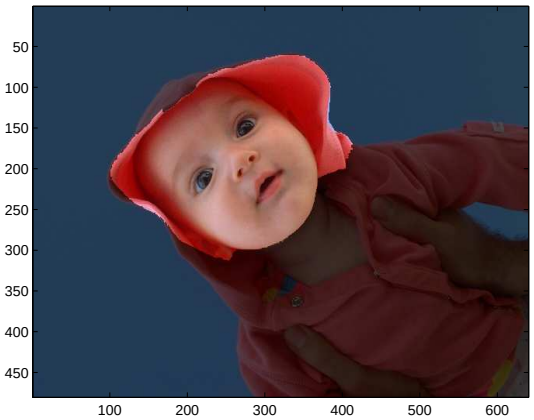
```
>> w = V(:,3);
>> S = sparsecut(a,w);
>> imcut(img,S)
```




```
>> w = V(:,4);
>> S = sparsecut(a,w);
>> imcut(img,S)
```




```
>> w = V(:,5);
>> S = sparsecut(a,w);
>> imcut(img,S)
```



I don't claim that these cuts are ideal. But, if you measured quality of cut per line of matlab, these are clear wins.